High Level Assembler for MVS & VM & VSE

Toolkit Feature User's Guide

Release 5

GC26-8710-08

High Level Assembler for MVS & VM & VSE

Toolkit Feature User's Guide

Release 5

GC26-8710-08

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 327.

Ninth Edition (June 2004)

This edition applies to IBM High Level Assembler for MVS & VM & VSE, Release 5, Program Number 5696-234 and to any subsequent releases until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
J87/D325
555 Bailey Avenue
SAN JOSE, CA 95141-1003
United States of America

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1992, 2004. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book v
Who Should Use This Book v
Syntax Notation v

Summary of Changes ix
Disassembler ix
Enhanced SuperC ix
Interactive Debug Facility ix

Chapter 1. Toolkit Feature Introduction 1
Toolkit Feature Components 2
Potential Uses for the Toolkit Feature 6

Chapter 2. Using Structured Programming Macros 11
Introduction to Structured Programming Macros 11
Accessing the Macros 12
The ASMMREL Macro 13
The IF Macro Set 14
The DO Macro Set 21
The SEARCH Macro Set 31
The CASE Macro Set 34
The SELECT Macro Set 37

Chapter 3. Using the Disassembler 41
Introduction to the Disassembler 41
Invoking the Disassembler 42
Control Statements 48
Disassembling a Module for the First Time 52
Output Description 53
Disassembler CMS Messages 55
Disassembler Messages 57

Chapter 4. Using the Program Understanding Tool 61
Introducing ASMPUT 62
Working with ADATA files 67
Working with the control flow graph 74
ASMPUT windows and window areas 91
Restrictions 95
Using online help 96
ASMPUT messages 97

Chapter 5. Using the Cross-Reference Facility 105
Introduction to ASMXREF 106
Invoking the Cross-Reference Facility 108
ASMXREF Control Statements 125
ASMXREF Token Statement 128
ASMXREF Options 132
ASMXREF XRFLANG Statements 132
ASMXREP Options 135
Understanding the Reports 135

© Copyright IBM Corp. 1992, 2004

ASMXREF Messages 158
ASMXREF User Abends 169

Chapter 6. Using Enhanced SuperC 171
Introduction to Enhanced SuperC 174
How SuperC and Search-For Filter Input File Lines 177
How SuperC Corrects False Matches 178
How SuperC Partitions and Processes Large Files 179
Comparing Load Modules 179
Comparing CSECTs 179
Invoking the SuperC Comparison 180
Invoking the SuperC Search 205
Process Options 226
Process Statements 236
CMS Command Line Option Directives 266
CMS Command Line Statement Option Directives 267
Understanding the Listings 269
Update Files 292
CMS File Selection List 305
How SuperC Pairs CMS Files and Members 311
CMS Files Used by SuperC 312
Reasons for Differing Comparison Results 313
Return Codes 314
SuperC Messages 316

Notices 327

Glossary 329

Bibliography 337
High Level Assembler Publications 337
Toolkit Feature Publications 337
Related Publications (Architecture) 337
Related Publications for MVS 337
Related Publications for VM 338
Related Publications for VSE 338

Index 339

Windows is a trademark of Microsoft Corporation. Windows NT is a registered trademark of Microsoft Corporation.

About This Book

This book describes how to use the following components of the High Level Assembler Toolkit Feature:

- Structured Programming Macros
- Disassembler
- Program Understanding Tool (ASMPUT)
- Cross-Reference Facility (ASMXREF)
- Enhanced SuperC

Throughout this book, we use these indicators to identify platform-specific information:

- Prefix the text with platform-specific text (for example, “Under CMS...”)
- Add parenthetical qualifications (for example, “(CMS only)”)
- Bracket the text with icons. The following are some of the icons that we use:

  MVS Informs you of information specific to MVS™ MVS

  CMS Informs you of information specific to CMS CMS

  VSE Informs you of information specific to VSE VSE

| MVS is used in this manual to refer to Multiple Virtual Storage/Enterprise Systems
| Architecture (MVS/ESA™), to OS/390®, and to z/OS®.

| CMS is used in this manual to refer to Conversational Monitor System on z/VM®.

| VSE is used in this manual to refer to Virtual Storage Extended/Enterprise Systems
| Architecture (VSE/ESA™), and z/VSE.

Who Should Use This Book

This book is for programmers who code in the High Level Assembler language or wish to use a component of the HLASM Toolkit Feature.

To use this book, you need to be familiar with the High Level Assembler language, the MVS, VM, or VSE operating system, the publications that describe your system, and job control language (JCL) or EXEC processing.

Syntax Notation

Throughout this book, syntax descriptions use the structure defined below.

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

  The ►►— symbol indicates the beginning of a statement.

  The —► symbol indicates that the statement syntax is continued on the next line.

  The ►— symbol indicates that a statement is continued from the previous line.

  The —►◄ indicates the end of a statement.

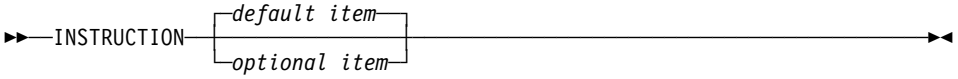
© Copyright IBM Corp. 1992, 2004

Diagrams of syntactical units other than complete statements start with the ►— symbol and end with the —► symbol.

- **Keywords** appear in uppercase letters (for example, ASPACE) or upper and lower case (for example, PATHFile). They must be spelled exactly as shown. Lower case letters are optional (for example, you could enter the PATHFile keyword as PATHF, PATHFI, PATHFIL or PATHFILE).

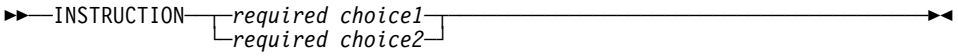
  Variables appear in all lowercase letters in a special typeface (for example, *integer*). They represent user-supplied names or values.
- If punctuation marks, parentheses, or such symbols are shown, they must be entered as part of the syntax.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path. If the item is optional and is the default, the item appears above the main path.

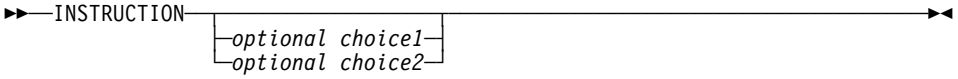


- When you can choose from two or more items, they appear vertically in a stack.

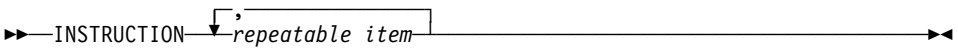
  If you **must** choose one of the items, one item of the stack appears on the main path.



  If choosing one of the items is optional, the whole stack appears below the main path.



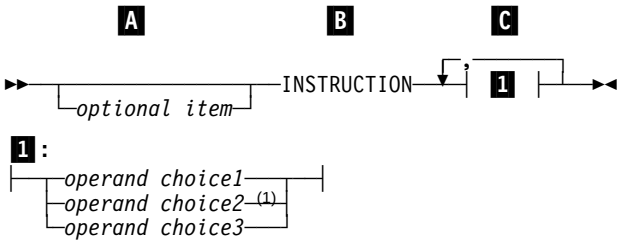
- An arrow returning to the left above the main line indicates an item that can be repeated. When the repeat arrow contains a separator character, such as a comma, you must separate items with the separator character.



  A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.

The following example shows how the syntax is used.

Format



Note:
[1] *operand choice2* and *operand choice3* must not be specified together

A The item is optional, and can be coded or not.

B The INSTRUCTION key word must be specified and coded as shown.

C The item referred to by **1** is a required operand. Allowable choices for this operand are given in the fragment of the syntax diagram shown below **1** at the bottom of the diagram. The operand can also be repeated. That is, more than one choice can be specified, with each choice separated by a comma.

Summary of Changes

Date of Publication June 2004

Form of Publication Ninth Edition, GC26-8710-08

This manual has been amended to incorporate the following technical changes:

Disassembler

- **New options:**

  NEWNUM Allows numeric fields within control statements to be entered as either hexadecimal or decimal values.

  OPTABLE Allows the operation code table to be used in disassembling the CSECT to be specified.

  VSESVC Uses the VSE description for SVCs, not the MVS description. Used to disassemble VSE code while running on MVS.

Enhanced SuperC

- **New process option:**

  SYSIN Allows an alternate DD name to be specified for process statements (MVS only).

- **New process statements:**

  NEWDD Allows alternative names to be specified for the new file on MVS.

  OLDDD Allows alternative names to be specified for the old file on MVS.

  UPDDD Allows alternative names to be specified for the output update file on MVS.

  WORKSIZE Allows the maximum size of the comparison set to be adjusted for comparing large files.

Interactive Debug Facility

- LUNAME now supported on MVS.
- In addition to a native TSO session, IDF can be run in either a TSO batch job or an MVS batch job if VTAM is used for the IDF terminal session.

© Copyright IBM Corp. 1992, 2004

Chapter 1. Toolkit Feature Introduction

The High Level Assembler Toolkit Feature is an optional, separately priced feature of IBM High Level Assembler for MVS & VM & VSE* (HLASM*). It provides a powerful and flexible set of tools to improve application recovery and development on OS/390*, MVS/ESA*, VM/ESA*, and VSE/ESA* systems. These productivity-enhancing tools are:

Structured Programming Macros
: A complete set of macro instructions that implement the most widely used structured-programming constructs (IF, DO, CASE, SEARCH, SELECT). These macros simplify coding and help eliminate errors in writing branch instructions.

Disassembler
: A tool which converts binary machine language to assembler language source statements. It helps you understand programs in executable (object or “load”) format, and enables recovery of lost source code.

Program Understanding Tool
: A workstation-based program analysis tool. It provides multiple and “variable-magnification” views of control flows within single assembled programs or across entire load modules.

Cross-Reference Facility
: A flexible source code cross-referencing tool. It helps you determine variable and macro usage, and locates specific uses of arbitrary strings of characters.

Interactive Debug Facility
: A powerful and sophisticated symbolic debugger for applications written in assembler language and other compiled languages. It simplifies and speeds the development of correct and reliable applications. For details on the Interactive Debug Facility see the *HLASM Toolkit Feature Interactive Debug Facility User's Guide* and the *HLASM Toolkit Feature Debug Reference Summary*.

Enhanced SuperC
: A versatile comparison and search facility for comparing two sets of data and showing the differences in an easy-to-read format.

  Special features allow for the “smart comparison” of dates.

  You can exclude certain data from the comparison.

  Enhanced SuperC also provides an extensive search tool.

Together, these tools provide a powerful set of capabilities to speed application development, diagnosis, and recovery.

The following sections describe these components in three phases. Each phase is typical of program development, maintenance, conversion, and enhancement activities such as:

- Recovery and reconstruction of symbolic assembler language source code
- Analysis and understanding of complex assembler language programs
- Modification and testing

© Copyright IBM Corp. 1992, 2004

Toolkit Feature Components

First, a description of each of the components. Note that you can use the Toolkit Feature's components independently of HLASM.

Toolkit Feature Structured Programming Macros

The HLASM Toolkit Structured Programming Macros simplify the coding and understanding of complex control flows, and help to minimize the likelihood of introducing errors when coding test and branch instructions. For details on Structured Programming Macros see Chapter 2, “Using Structured Programming Macros” on page 11.

These macros support the most widely used structured-programming control structures and eliminate the need to code most explicit branches.

You can use the Toolkit Feature Structured Programming Macros to create the following structures:

IF/ELSE/ENDIF
: One-way or two-way branching, depending on simple or complex test conditions.

DO/ENDDO and STRTSRCH/ORELSE/ENDLOOP/ENDSRCH
: A rich and flexible set of looping structures with a variety of control and exit facilities.

CASENTRY/CASE/ENDCASE
: Fast N-way branching, based on an integer value in a register. Deciding which branch to take is made at the CASENTRY macro; a direct branch to the selected CASE is then done, followed by an exit at the ENDCASE macro.

  There is no OTHRWISE facility within this macro set.

SELECT/WHEN/OTHRWISE/ENDSEL
: Sequential testing, based on sets of comparisons. These macros create a series of tests that are evaluated in the order they are specified in the program. If a test is true, the WHEN section of code for that test is executed, followed by an exit at the ENDSEL macro.

  If no test is satisfied, then the OTHRWISE section (if present) is performed.

All the macro sets may be nested, and there are no internal limits to the depth of nesting. Tests made by the various END*xxx* macros ensure that each structure's nesting closure is at the correct level, and diagnostic messages (MNOTEs) are issued if they are not.

Toolkit Feature Disassembler

The HLASM Toolkit Feature Disassembler lets you extract single control sections (CSECTs) from object modules or executables such as load modules and phases. It converts them to assembler language statements that you can assemble to generate the same object code. For details on the Disassembler see Chapter 3, “Using the Disassembler” on page 41.

Your first control statement specifies the module and control section you are to disassemble. Adding control statements provides further guidance and helpful information to the Disassembler, allowing it to create a more readable program. You can supply sets of control statements in the primary input stream to the Disassembler, or (as each set is developed) you can save them in a library and direct the Disassembler to read them using COPY control statements.

- You can describe the layout of the control section with control statements asserting that certain areas of the module contain data only, instructions only, or are known to be uninitialized.
- You can request symbolic resolutions of halfword base-displacement storage by supplying control statements giving base addresses and the base registers for addressing.
- You can define data structures (DSECTs) and assign your own labels to designated positions in the program.
- The Disassembler automatically assigns symbolic names to registers. Branch instructions use extended mnemonics where possible, and identifies supervisor call (SVC) instructions when known.
- The Disassembler listing provides a full summary of the inputs and outputs of the disassembly, and places the reconstructed assembler language source program in a separate PUNCH file.

If you use the High Level Assembler with the ADATA option to assemble disassembler-generated statements, High Level Assembler generates a SYSADATA file (sometimes called the ADATA file). You can use this file as input to other Toolkit Feature components. This combination of facilities can help you recover lost source code written in *any* compiled language.

Toolkit Feature Program Understanding Tool

The Program Understanding tool (ASMPUT) helps you analyze and extract information about assembler language applications, using an OS/2® or a Windows graphical user interface to display graphical and textual views of an application's structure. ASMPUT extracts application analysis information from the SYSADATA file generated during host assembly by HLASM; you must download this ADATA file to your workstation for analysis. For details on Program Understanding Tool see Chapter 4, “Using the Program Understanding Tool” on page 61.

You can use ASMPUT to display selected programs and modules in these linked views:

- A Content view
- An Assembled Listing view
- A graphical Control Flow view
- An Expanded Source Code view

These views provide complete high-level to low-level information about assembler language applications:

- At the highest level, you can discover the relationships among programs and within modules.
- You can gradually descend program layers discovered by analysis of the individual programs to arrive at the lowest level, where you can examine details of internal control flows within each program.

ASMPUT lets you display multiple views of a given program or module. These multiple views are linked: scrolling through one view automatically scrolls through all other open views of that program, module, or application. Linked views help you see quickly the association between the assembled source code and the graphical control-flow representations of the program.

At any time, you can narrow or expand the focus of your analysis by zooming in or out on areas of particular interest. For example, you can use the View Contents window to scroll through the contents of an application and simultaneously see the change in control flow information displayed in the View Control Flow window.

ASMPUT displays several folders which provide a complete inventory of application analysis information, program samples, tools, documentation, help files, and a detailed tutorial to help you learn to use ASMPUT to analyze assembler language applications.

Toolkit Feature Cross-Reference Facility

The High Level Assembler Toolkit Cross-Reference Facility (ASMXREF) supports your maintenance tasks by scanning assembler language source, macro definitions, and copy files for symbols, macro calls, and user-specified tokens. For details on ASMXREF see Chapter 5, “Using the Cross-Reference Facility” on page 105.

You can use ASMXREF for identifying fields of application importance such as `DATE`, `TIME`, and `YYMMDD`. You can use an arbitrary “match anything” character (sometimes called a wildcard character) to create generic tokens such as “`"YY*"`”; the scan then searches for occurrences of the token with any other characters allowed in the position of the arbitrary character. You may also specify tokens to be *excluded* from a generic search, so that an exclude token such as “`SUMMER`” rejects matches of `SUMMER` when the include token is `*MM*`.

ASMXREF scans source code, in the following languages, for user-specified and default tokens:

- Assembler
- C
- C++
- COBOL
- FORTRAN
- PL/I
- REXX

ASMXREF provides several reports:

Control Flow (CF) Report

The CF report tabulates all intermodule program references as a function of member or entry point name, and lists them in the order of the members *referring to* the subject entry point or the entry point names *referred by* the subject member.

Lines of Code (LOC) Report

Provides a count, arranged by part and by component, of the number of source lines and comments in the part, and the shipped source instructions (SSI), which are the number of instructions within each part scanned, both executable and non-executable, that are not spaces or comments. As well, the report shows the changed source instructions (CSI), which are the number of unique

SSI that have been modified in each part categorized by added, changed, deleted, moved, and so on. In addition, the LOC Report provides a summary report of CSI arranged by programmer.

Lines of OO Code (LOOC) Report
Provides, for C++, the Lines of Code (LOC) per Class and per Object, and Objects per Class.

Macro Where Used (MWU) Report
Lists all macros invoked and all segments copied, including the type and frequency of the invocation or reference.

Symbol Where Used (SWU) Report
Lists all symbols referenced within the source members, and the type of reference. These symbols can be variables or macros.

Spreadsheet Oriented (SOR) Report
A comma-delimited file suitable for input into a standard spreadsheet application. It shows for each module scanned the number of lines of code, the number of occurrences of each token, and the total number of token matches. This information helps you identify the critical modules in an application and estimate the effort required for modifications.

Token Where Used (TWU) Report
Contains similar information to the SOR report, but in an easily readable format.

Before ASMXREF generates the TWU report, it creates a Tagged Source Program (TSP). This program contains special inserted comment statements where tokens are found, so that subsequent assembly of the “tagged” file helps you track important variables during control-flow analysis using ASMPUT, see “Toolkit Feature Program Understanding Tool” on page 3.

Toolkit Feature Interactive Debug Facility

The HLASM Toolkit Feature Interactive Debug Facility (IDF) supports a rich set of capabilities that speed error detection and correction. Although IDF is intended primarily for debugging assembler language programs on MVS, VM, and VSE, you can also use it to debug programs written in most high-level languages.

- IDF provides multiple selectable views of a program, including separate windows for address stops, breakpoints, register displays, object code disassembly, storage dumps, language-specific support, register histories, non-traced routines, and other information. You can use these views in any order or combination.
- You can control execution of a program by stepping through individual instructions or between selected breakpoints or routines.
- If source code is available (which is almost always the case for programs assembled with HLASM), IDF can display source statements as it executes the program.
- The power of IDF is greatly magnified by its ability to pass control from any breakpoint to user exit routines written in REXX or other languages that can capture and analyze program data, and respond dynamically to program conditions.
- You can count instruction executions, and IDF can maintain an instruction execution history.

- You can dynamically modify storage areas and register contents during debugging by simply typing new values on the displays.
- IDF supports a special class of conditional breakpoints called watchpoints, which IDF triggers only when a user-specified condition occurs.
- A command-level record and playback facility allows a debugging session to be re-executed automatically.
- Extensive tailoring capabilities allow you to establish a familiar debugging environment. Most debugging actions can be easily controlled by PF-key settings.

For more details on Interactive Debug Facility see the *HLASM Toolkit Feature Interactive Debug Facility User's Guide* and the *HLASM Toolkit Feature Debug Reference Summary*.

Enhanced SuperC

The Enhanced SuperC (known as SuperC) is a versatile comparison and search facility that can be used to compare two sets of data (using the SuperC Comparison) or to search a specific set of data for a nominated search string (using the SuperC Search).

At a minimum, the SuperC Comparison requires only the names of the two items to be compared. The SuperC Search requires only the name of the item to be searched and the search string.

You can tailor the comparison or search using *process options* and *process statements*. Process options are single keywords that you enter on the PARM parameter (MVS and VSE), a menu (CMS), or the command line (CMS). Process statements consist of a keyword and one or more operands; you pass these to SuperC in an input file.

For example, you can use the *process option* ANYC (“Any Case”) so that SuperC treats uppercase and lowercase characters as the same. (Thus, “d” and “D” are considered to be the same.) You can use the *process statement* DPLINE (“Do not Process Lines”) to ignore the lines (being compared or searched) that contain a specified character string. For example, `DPLINE '$'` causes all lines that contain the single-character string “$” to be ignored.

Potential Uses for the Toolkit Feature

The following figure shows three phases of a redevelopment project. This section describes potential uses for the Toolkit Feature during each of these phases.

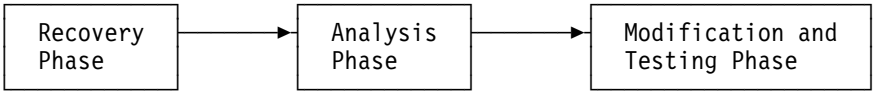


Figure 1. Typical Phases for Toolkit Feature Usage

1. ***Recovery and reconstruction*** of assembler language source statements from object modules, or load modules, for which the original source is lost. The disassembler initially produces non-symbolic assembler language source from

object code. You can add control statements iteratively to help define code, data, USINGs, labels, and DSECTs symbolically.

2. ***Analysis and understanding*** of assembler language source programs can benefit from three Toolkit components: the Cross-Reference Facility, the Program Understanding Tool, and the Interactive Debug Facility.

   a. You can use the Cross-Reference Facility token scanner to locate important symbols, user-selected tokens, macro calls, and other helpful data. ASMXREF also creates an “impact-analysis” file for input to a spreadsheet application for effort estimation and impact assessment. Another ASMXREF output is a Tagged Source Program: when assembled with the ADATA option, this program produces a SYSADATA file for you to use with the Program Understanding Tool.

   b. The Program Understanding Tool provides graphic displays of program structure, control flow, a simplified listing, and other views with any desired level of detail. With the ADATA file created from the tagged source produced by ASMXREF, you can rapidly locate and analyze key areas of the program.

   c. The Interactive Debug Facility is by design a “program understanding” tool that lets you monitor the behavior of programs at every level of detail. You can monitor and trace data flows among registers and storage, even showing the operations of individual instructions!

   You can use the Disassembler, Cross-Reference Facility, Program Understanding Tool and Interactive Debug Facility together to help reconstruct lost assembler language source (with the same function as that produced by a high level language compiler).

3. ***Modification and Testing*** of updated programs is simplified by using the powerful Interactive Debug Facility. At the same time, you can simplify program logic by replacing complex test/branch logic with the Structured Programming Macros.

   You can use the Enhanced SuperC to compare an original source file with a modified source file, or a pre-migration application output file with a post-migration output file, and report the differences between the files. Enhanced SuperC can report all differences, or you can set options to exclude the reporting of differences when those differences are correctly-modified date fields. You can also limit the comparison to date fields only.

Recovery and Reconstruction

During the Recovery and Reconstruction phase, you typically begin with a program in object or executable format. Using the Disassembler, and by providing suitable control statements, you can create an assembler language source program with as much structure and symbolic labeling as you like.

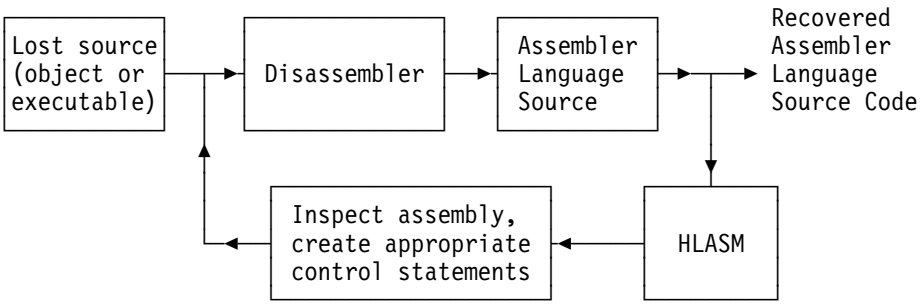


Figure 2. Toolkit Feature: Recovery and Reconstruction Phase

Repeat the disassembly/analysis/description/assembly cycle until you obtain satisfactory assembler language source code.

The initial steps do not require reassembly of the generated assembler language source, as appropriate control statements are usually easy to determine from the Disassembler's listing. As the recovered program approaches its final form, you should assemble it with HLASM to ensure the validity of your new source program.

Analysis and Understanding

The most complex aspect of application maintenance and migration is analyzing and understanding the code. There are three components of Toolkit Feature that can help:

ASMXREF Can locate all uses of a variable name or any character string. You can also produce a Tagged Source Program.

ASMPUT Provides graphical views of control flows within and among programs and modules.

IDF Helps you monitor and track the behavior of individual instructions and data items.

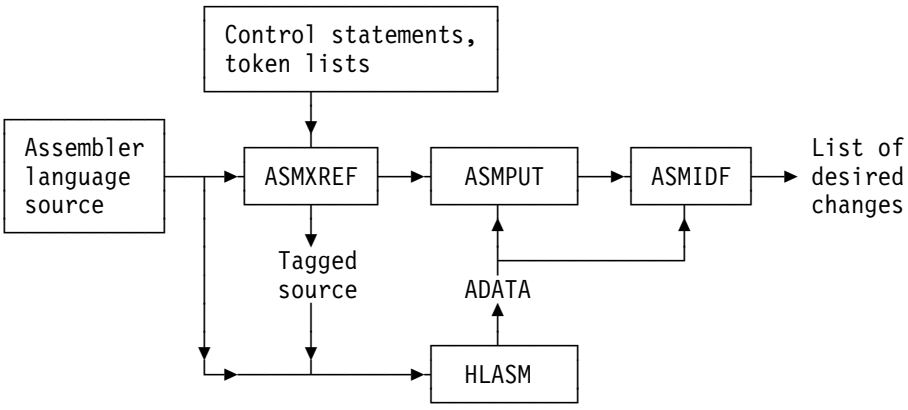


Figure 3. Toolkit Feature: Analysis and Understanding Phase

While each of these components has valuable capabilities, using them in combination can provide great synergy in analyzing and understanding program behavior.

Modification and Testing

After you have used the Disassembler, ASMXREF, and ASMPUT components to determine the needed modifications, you can add Structured Programming Macros to simplify the coding and logic of the program.

You can then test the updated code using the rich and flexible features of the Interactive Debug Facility. After each assembly/debug cycle, you can further modify the source code, repeating the process until the completed application is accepted for installation in a production library. You can use Enhanced SuperC to compare the original source with the modified source, checking that all references to date have been correctly modified.

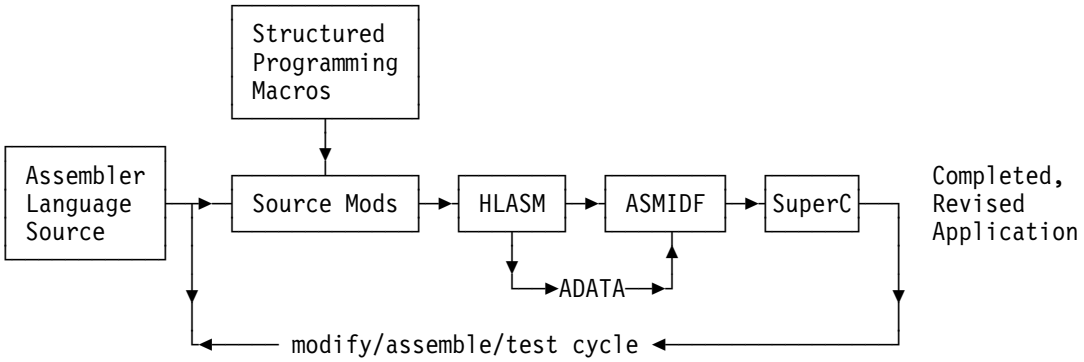


Figure 4. Toolkit Feature: Modification and Testing Phase

Summary

These phases illustrate how the HLASM Toolkit Feature provides a varied and powerful set of tools supporting all aspects of application development, maintenance, enhancement, and testing. The following figure summarizes these capabilities:

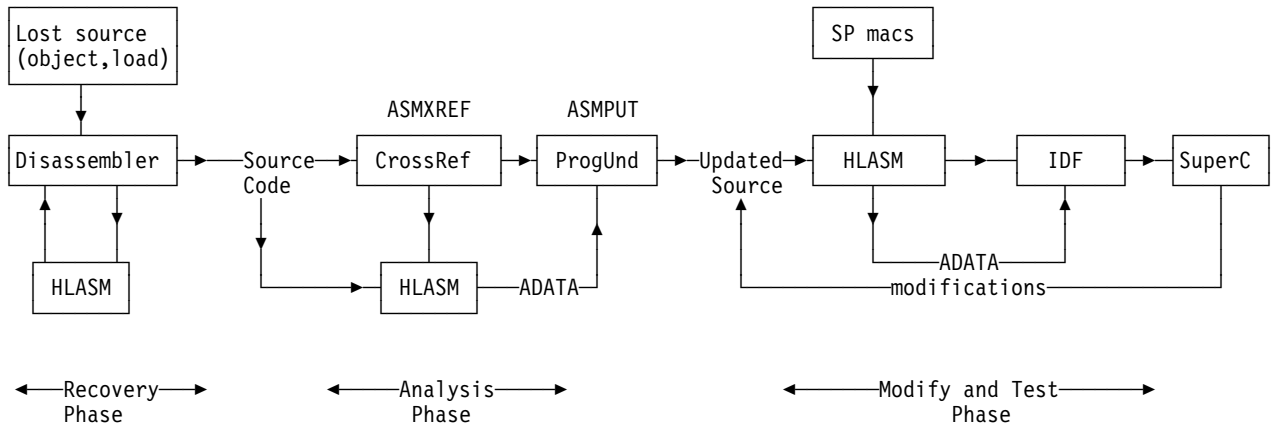


Figure 5. Toolkit Feature: Summary of Usage Phases

A typical process for managing the full spectrum of application recovery, development, and maintenance activities includes several steps. Table 1 on page 10 shows the Toolkit Feature tools useful each step.

Table 1. Toolkit Feature Components

Activity	Toolkit Feature Components
Inventory and assessment	The **Disassembler** can help recover programs previously unretrievable or unmodifiable.
Locate date fields and uses	The **Cross-Reference Facility** pinpoints date fields and localizes references to them in single or multiple modules. **SuperC** provides powerful string-search facilities.
Application understanding	The **Program Understanding Tool** provides powerful insights into program structures and control flows. The **Interactive Debug Facility** monitors instruction and data flows at any level of detail.
Decide on fixes and methods	
Implement changes	The **Structured Programming Macros** clarify source coding by reducing the need for coding branches and tests, replacing them with readable structures. **SuperC** helps verify that source changes are complete.
Unit test	The **Interactive Debug Facility** provides powerful debugging and tracing capabilities for verifying the correctness of changes.
Debug	The **Interactive Debug Facility** helps debug complete applications, including dynamically loaded modules.
Validation	**SuperC** checks regressions, validates correctness of updates.

Chapter 2. Using Structured Programming Macros

Introduction to Structured Programming Macros . . . 11
Accessing the Macros . . . 12
The ASMMREL Macro . . . 13
The IF Macro Set . . . 14
IF Macro Option A . . . 15
IF Macro Option B . . . 16
IF Macro Option C . . . 16
IF Macro Option D . . . 17
IF Macros with Boolean Operators . . . 18
The ELSEIF Macro . . . 20
The DO Macro Set . . . 21
The DO Indexing Group . . . 22
DO Loop Terminator Generation . . . 22
Infinite Loop . . . 23
Branching To the ENDDO . . . 23
Leaving a Nested DO . . . 25
Explicit Specification . . . 26
Counting . . . 27
Backward Indexing . . . 27
Forward Indexing . . . 28
Register Initialization . . . 29
The UNTIL and WHILE Keywords . . . 29
The SEARCH Macro Set . . . 31
The CASE Macro Set . . . 34
The SELECT Macro Set . . . 37

Introduction to Structured Programming Macros

The complexity of control flow in a program strongly affects its readability, the early detection of coding errors, and the effort needed to modify it later. You can usually simplify control flow (though sometimes at the cost of less efficiency and more redundant code) by restricting the ways in which branches occur. One way to restrict branches is to use only those necessary to implement a few basic structures such as:

- Executing one of two blocks of code according to a true-false condition
- Executing a block of code repeatedly until some limit is reached
- Executing a specific block of code, in a given set, where the block was previously computed

If statements exist for all these structures in a programming language, then they are used exclusively. If some are missing, then simple branches are used to simulate those structures but only in standard patterns. In the case of OS assembler language, only the basic branch and branch-and-link instructions are implemented but macros that simulate the first three structures are available.

The first two structures are sufficient to implement any “proper” program (that is, with one entry point and one exit) provided that its blocks of code are suitably ordered. It is assumed that the structures may be nested to any depth. The

© Copyright IBM Corp. 1992, 2004

technique of writing programs using only these structures for branching is known as "structured programming".

The standard structured programming figures have been implemented for the assembler language programmer through the following five sets of related macros.

- The IF macro set:

  IF
  ELSE (optional)
  ELSEIF (optional)
  ENDIF

- The DO macro set:

  DO
  DOEXIT (optional)
  ITERATE (optional)
  ASMLEAVE (optional)
  ENDDO

- The CASE macro set:

  CASENTRY
  CASE (one must be present)
  ENDCASE

- The SEARCH macro set:

  STRTSRCH
  EXITIF
  ORELSE
  ENDLOOP
  ENDSRCH

- The SELECT macro set:

  SELECT
  WHEN
  OTHRWISE (optional)
  ENDSEL

- The ASMMREL macro set:

  ASMMREL

Accessing the Macros

To use these macros:

- Ensure the macro library provided as part of the Toolkit Feature is included; for MVS, in the SYSLIB concatenation; for CMS, in the GLOBAL MACLIB command; or for VSE, in the LIBDEF SOURCE search chain.

  For MVS, the default SMP/E target library is *hlq*.SASMMAC2.

  For VSE, the default sublibrary is PRD2.PROD.

  For CMS, the default location is *userid* P696234H disk 29E macro library ASMSMAC MACLIB.

- Add the following statement to the program:

```
COPY  ASMMSP
```

Add this statement prior to any line containing a macro. You can add this statement either directly by updating the actual file or by using the PROFILE facility of High Level Assembler. This COPY statement must be inserted before any use is made of these macros.

The ASMMSP member contains all the macros described here. This has been done to avoid name collisions with any existing macros. If there is a collision, then use OPSYN statements following the COPY statement to rename macros using duplicate names. Any use of these macros must refer to the new names.

The following restrictions apply when using these macros:

- The macros generate labels of the following format:

  `#@LB`*n* `DC 0H`

  where *n* is a sequence number starting at 1.

  Do not use these names for any labels within the user's program.

- The macros use a set of global macro variables for processing. The definitions for these variables are in ASMMGBLV (this is a member in the supplied library). These macro variable names must not be used in any other macros.
- The following words are reserved keywords and must not be used for operands or instructions: AND, OR, ANDIF, ORIF.
- It is strongly suggested that you do not use the mnemonic keywords in Table 3 on page 15 as labels or operands.

The ASMMREL Macro

By default, the structured programming macros generate base branch on condition instructions. You can get the macros to generate branch relative on condition instructions using the ASMMREL macro.

The ASMMREL macro can be used as follows:

```
ASMMREL  ON
```

This macro sets a global variable that causes all subsequent macro expansions to use branch relative instructions. The operand is optional and the default is ON. To revert to using base displacement branches then insert the following statement in the program:

```
ASMMREL  OFF
```

Using ASMMREL causes the following:

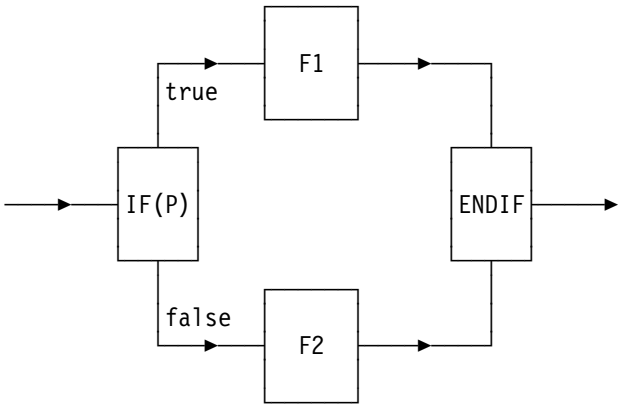
1. The DO macro generates LHI where it can to initialise registers.
2. DO loop terminator is generated according to:

   ```
   ASMMREL OFF - one of: BC  BXH  BXLE  BCT  BCTR
   ASMMREL ON  - one of: BRC BRXH BRXLE BRCT BCTR
   ```

3. The CASENTRY macro generation alters the contents of R0.

The IF Macro Set

The IF macro set implements the IF THEN ELSE program figure. The flowchart for this figure is:



In this figure, the test of the predicate *p* is represented by the IF macro, which determines whether process F1 or F2 is to be executed. The exit path from the macro is represented by the terminator ENDIF macro. The general IF macro set is written:

In the IF examples that follow, the parentheses surrounding the predicate are optional.

```
IF  p  THEN
  Code for F1
ELSE
  Code for F2
ENDIF
```

If the ELSE is not used, the flowchart is reduced to one that does not contain function F2 and is written:

```
IF p THEN
  Code for F1
ENDIF
```

The format of the predicate *p* may take one of the forms discussed in Table 2. In each form the keywords AND, OR, ANDIF, and ORIF are optional. THEN is a comment and must be preceded by one or more spaces if used.

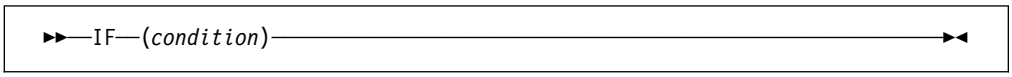
All of these forms of the predicate *p* may be used in the DOEXIT, EXITIF, and WHEN macros.

Predicate Values	Connector/Terminator
numeric condition code (1 to 14) condition mnemonic instruction, parm1, parm2, condition compare-instruction, parm1, condition, parm2	AND OR ANDIF ORIF

Table 2. Predicate values and Connector/Terminator Values

Note: Do not use the connectors AND, OR, ANDIF, and ORIF as program labels.

IF Macro Option A



Option A tests the previously set condition code. It uses the Extended Branch Mnemonics for the branch instruction or the numeric condition code masks to indicate the condition. Table 3 following the examples shows the mnemonics and their complements.

```
        IF  (H)  THEN
          Code for F1
         ELSE
          Code for F2
         ENDIF
```

generates this code:

```
         IF  (H)  THEN
           BC           15-2,#@LB1
           Code for F1
         ELSE
             BC             15,#@LB3
#@LB1                  DC  0H
           Code for F2
         ENDIF
#@LB3                  DC  0H
```

The same example, using a numeric condition code mask, is:

```
         IF  (2)  THEN
           Code for F1
         ELSE
           Code for F2
         ENDIF
```

This generates the same code.

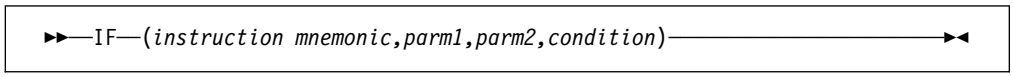
Case	Condition Mnemonics	Meaning	Complement
After compare instructions	H, GT L, LT E, EQ	high, greater than low, less than equal	NH, LE NL, GE NE
After arithmetic instructions	P M Z O	plus minus zero overflow	NP NM NZ NO
After test under mask instructions	O M Z	ones mixed zeros	NO NM NZ

Table 3. Mnemonics and Complements

Notes:

1. Do not use the mnemonics and complement symbols as program labels.
2. The mnemonics shown in the table can be in lowercase.

IF Macro Option B



Option B needs all four parameters.

The *instruction mnemonic* is any other than a compare, that sets the condition code. (Use option A if the condition code has been set previously.)

The parameters *parm1* and *parm2* are the two fields associated with the instruction.

Condition is the value that the condition code mask must assume for the THEN clause to be executed. The *condition* parameter is either one of the condition mnemonics given in Table 3 on page 15, or a numeric condition code.

This example of option B:

```
         IF  (TM,BYTE,X'80',Z)  THEN
           Code for F1
         ELSE
           Code for B2
         ENDIF
```

generates:

```
         IF  (TM,BYTE,X'80',Z)  THEN
                TM              BYTE,X'80'
           BC           15-8,#@LB1
           Code for F1
         ELSE
             BC             15,#@LB3
#@LB1                  DC  0H
           Code for B2
         ENDIF
#@LB3                  DC  0H
```

Option B also provides for three-operand instructions such as those that are available on the System/370. For example:

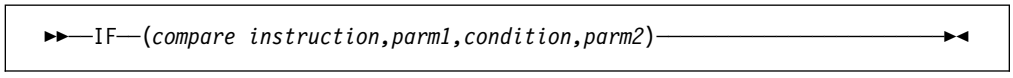
```
         IF    (ICM,R1,M3,B2(D2),4)
```

generates:

```
         ICM  R1,M3,B2(D2)
         BC   15-4,L1
```

In all option B formats, the instruction is coded first, followed by the appropriate operands in the same order as used in open code, and followed by the *condition* operand.

IF Macro Option C



Option C needs all four parameters.

Any compare instruction is valid. However, with a compare instruction, the condition mnemonic appears between *parm1* and *parm2*, instead of after both of them as in option B.

In all cases, *parm1* and *parm2* must agree, as if you were writing the instruction in assembler language.

The *condition* parameter is either condition mnemonic from Table 3 on page 15, or a numeric condition code.

This example of option C:

```
         IF  (CLI,0(2),EQ,X'40')  THEN
           Code for F1
         ELSE
           Code for F2
         ENDIF
```

generates:

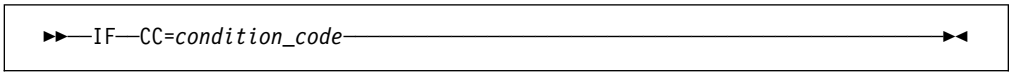
```
         IF  (CLI,0(2),EQ,X'40')  THEN
                CLI             0(2),X'40'
           BC           15-8,#@LB1
           Code for F1
         ELSE
             BC              15,#@LB3
#@LB1                  DC  0H
           Code for F2
         ENDIF
#@LB3                  DC  0H
```

Option C also provides for three-operand compare instructions. An example is:

```
         IF  (CLM,R1,M3,NE,B2(D2))
```

In all option C formats, the instruction is coded first, followed by the appropriate operands in the same order as used in open code, with the condition code operand in the next to last position.

IF Macro Option D



Where:

condition_code Numeric condition code

Option D tests the previously set condition code. It uses the numeric condition code to indicate the condition.

The following example:

```
         IF CC=2  THEN
           Code for F1
         ELSE
           Code for F2
         ENDIF
```

generates the following:

```
          IF CC=2  THEN
            BC           15-2,#@LB1
            Code for F1
          ELSE
              BC             15,#@LB3
#@LB1                  DC  0H
            Code for F2
          ENDIF
#@LB3                  DC  0H
```

Note: This form of the IF macro cannot be used with Boolean Operators

IF Macros with Boolean Operators

All the options described in the preceding sections can be combined into longer logical expressions using Boolean operators AND, OR, ANDIF, and ORIF. (These are reserved keywords and cannot be used as operands of instructions.) A NOT operator has not been implemented since a complement exists for each of the alphabetic condition mnemonics described previously.

All logical expressions are scanned from left to right. When the AND and OR connectors are used, the code generated is such that as soon as the expression can be verified as either true or false the appropriate branch to process either the code for F1 or the code for F2 is taken without executing the remaining tests. Statements that are continued onto more than one line must have a non-space character in the continuation indicator column (usually column 72) of all statements except the last. Continued statements must have a non-space character in the continuation column (usually column 16)

This example:

```
         IF  (10),OR,                                                  X
             (AR,R2,R3,NZ),AND,                                        X
             (ICM,R1,M3,B2(D2),4)  THEN
           Code for F1
         ELSE
           Code for F2
         ENDIF
```

generates:

```
         IF  (10),OR, X'40')                                            X
               (AR,R2,R3,NZ),AND,                                       X
               (ICM,R1,M3,B2(D2),4)  THEN
                BC              10,#@LB2
                AR              R2,R3
                BC              15-7,#@LB1
                ICM R1,M3,B2(D2)
           BC           15-4,#@LB1
#@LB2                  DC  0H
           Code for F1
         ELSE
             BC             15,#@LB3
#@LB1                  DC  0H
           Code for F2
         ENDIF
#@LB3                  DC  0H
```

Note that if the condition code mask setting is 10 upon entering the IF code, the program immediately branches to the F1 code. If it is not 10, and if the next condition code setting is such that the desired relation is not true, the branch is made around the third test to the F2 code. This is done since the AND condition cannot be met if the second relation is false.

The ANDIF and ORIF are used to give a parenthetical grouping capability to the logical expressions. The use of either of these two as connectors of logical groupings, the use of AND or OR indicates a closing parenthesis on the preceding group and an opening parenthesis on the one following. Therefore, if the previous example is modified by replacing the AND by an ANDIF, this means that either the first or second condition must be true as well as the third one in order to execute F1.

An example of this:

```
         IF (10),OR,                                                  X
            (AR,R2,R3,NZ),ANDIF,                                      X
            (ICM,R1,M3,B2(D2),4)  THEN
           Code for F1
         ELSE
           Code for F2
         ENDIF
```

generates:

```
         IF (10),OR,                                                  X
               (AR,R2,R3,NZ),ANDIF,                                   X
               (ICM,R1,M3,B2(D2),4)  THEN
                BC              10,#@LB2
                AR              R2,R3
                BC              15-7,#@LB1
#@LB2           DC              0H
                ICM R1,M3,B2(D2)
           BC           15-4,#@LB1
           Code for F1
         ELSE
             BC              15,#@LB4
#@LB1                  DC  0H
           Code for F2
         ENDIF
#@LB4                  DC  0H
```

For a better illustration of the effect of the ANDIF and ORIF usage, the examples which follow use capital letters to indicate the conditions that are tested.

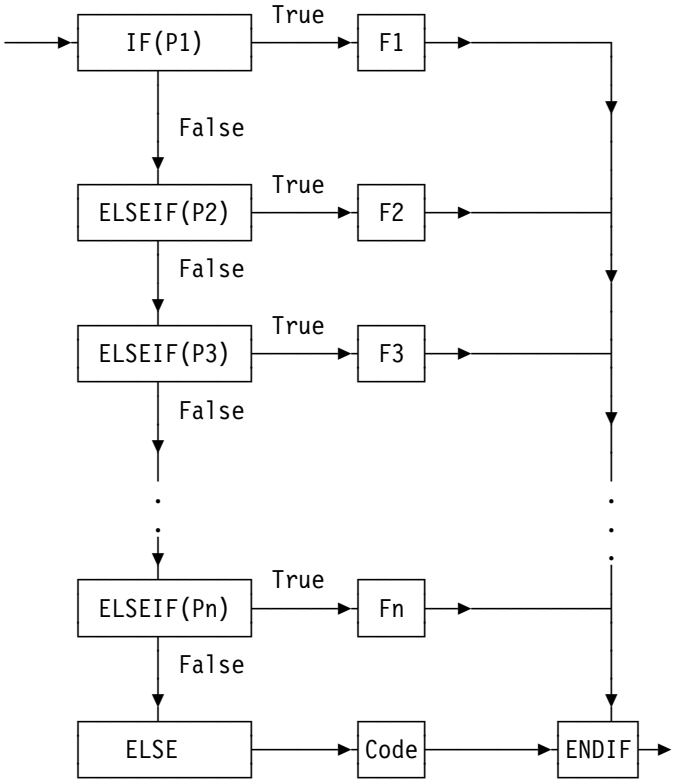
If you write: A OR B AND C, the implied grouping is A OR (B AND C).

If you write: A OR B ANDIF C, the grouping is (A OR B) AND C.

The ORIF may be similarly used: A AND B ORIF C OR D is interpreted as (A AND B) OR (C OR D).

The ELSEIF Macro

The ELSEIF macro is an optional part of the IF macro set. It provides the means for a series of checks, where a function is executed once the predicate condition has been satisfied. The flowchart for an IF including an ELSEIF is:



The predicate for the ELSEIF macro is one of the forms permitted for the IF macro.

This example:

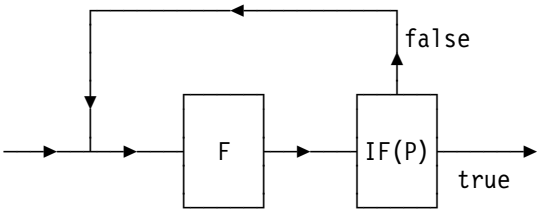
```
if (clc,a,eq,b)
 mvc a,d
elseif (clc,e,eq,f)
  mvc g,h
elseif (clc,g,eq,h)
  mvc i,k
endif
```

generates:

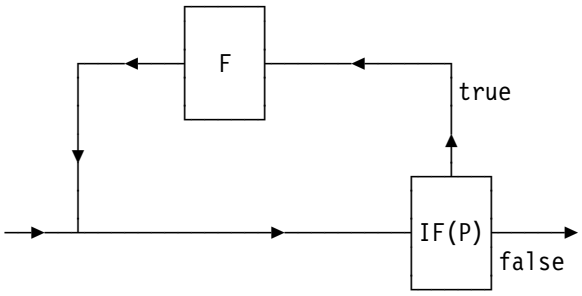
```
          if (clc,a,eq,b)
+               clc             a,b
+           BC          15-8,#@LB1
           mvc a,d
          elseif (clc,e,eq,f)
+       BC              15,#@LB3
+#@LB1                   DC  0H
+               clc             e,f
+           BC           15-8,#@LB4
            mvc g,h
          elseif (clc,g,eq,h)
+       BC              15,#@LB6
+#@LB4                   DC  0H
+               clc             g,h
+           BC           15-8,#@LB7
            mvc i,k
          endif
+#@LB7                   DC  0H
+#@LB6                   DC  0H
+#@LB3                   DC  0H
```

The DO Macro Set

The flowchart represented by this set depends on the keywords used with the predicate *p*. If the UNTIL or the indexing group of key words (FROM, TO, BY) is used, the flowchart is:



If the WHILE keyword is specified, the flowchart is:



The general DO macro set is written as:

```
DO  P
  Code for F
ENDDO
```

The DO macro accepts zero or one positional parameter and five possible keywords. The positional parameter may be INF, BXH, or BXLE. The keywords are FROM, TO, BY, WHILE, and UNTIL. The FROM, TO, and BY keywords form

an indexing group that specifies ranges and increments when indexing through a loop. They indicate loop termination tests, which are made after execution of the function code F, and the determination of whether to repeat the loop is made by one of the four indexing instructions: BXH, BXLE, BCT, or BCTR. If an indexing instruction is not given explicitly as a positional parameter, one is derived from the other values given. Infinite looping is also permitted through use of the INF positional parameter.

The function of the UNTIL keyword is similar to that of the loop terminator, except that the determination of whether to repeat the function code F depends upon the result of any condition code setting instruction. It may not be used with the indexing group. See also "The DO Indexing Group."

The WHILE keyword, on the other hand, generates a test prior to entering the function code of the loop. It may be used with either the indexing group or the UNTIL keyword to provide tests at both initiation and termination of the function code.

| The following combinations of keywords are valid with the DO macro:

| FROM, TO, BY
| FROM, TO, BY, WHILE
| WHILE, UNTIL

| In all cases, the structure must be terminated by the ENDDO macro.

The DO Indexing Group

The indexing group permits five types of counting and testing to be performed. Each different requirement for counting and testing has a corresponding set of keywords and values, and results in the generation of appropriate loop initialization and termination instructions. The five variations are described in the following paragraphs and are summarized in the table below. The tests to determine which variation is to be used are performed in the order described in Table 3 on page 15.

In the indexing group, each of the three keywords is permitted to indicate a register designation and an optional value. Thus, an indexing DO statement could appear as:

```
DO  FROM=(Rx,i),TO=(Ry+1,j),BY=(Ry,k)
```

if all keywords in the group were used.

The format of the keywords is keyed to the BXH and BXLE indexing instructions, and the restrictions on the use of these instructions are carried over into the macros. Therefore, if the BY register Ry is an even-numbered register, then the TO register must be Ry+1. If the BY register Ry is an odd-numbered register, then the TO register must be the same register, and hence the TO and BY values (j and k, respectively) must be identical.

DO Loop Terminator Generation

The table below summarizes the various instructions that are generated to terminate DO loops. The types of loops are discussed following the table, including examples.

Type	Keywords	Other conditions	Result
Infinite Loop	Neither FROM WHILE nor UNTIL	INF parameter	BC 15
Explicit Specification	FROM, plus TO and/or BY	BXH parameter BXLE parameter	BXH BXLE
Counting	FROM (only)	Two values Three values	BCT BCTR
Backward Indexing	FROM, TO and BY	FROM and TO numeric FROM value greater than TO value	BXH
Backward Indexing	FROM BY	BY numeric and less than zero	BXH
Forward Indexing	Any combination not covered in the above cases		BXLE

Table 4. DO Loop Terminator Generation

Infinite Loop

If you wish to execute a loop until some external terminating event takes place (for example, an end of file), then you may do so by specifying the INF positional parameter.

Thus, coding:

```
         DO  INF
           Code for F
         ENDDO
```

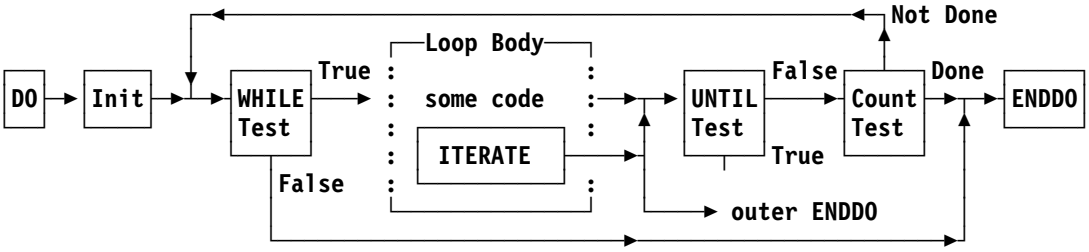
generates:

```
         DO  INF
#@LB2                  DC  0H
           Code for F
         ENDDO
                BC              15,#@LB2
```

In order to generate an infinite loop, no FROM, WHILE, or UNTIL keywords can be present. TO and BY keywords, if present, are ignored.

Branching To the ENDDO

The ITERATE macro causes a branch to the point prior to the ENDDO macro associated with the active DO macro. If a label is specified, then the ITERATE branches to the point prior to the ENDDO macro associated with the DO macro with the label. Here is the flowchart for this structure:



In the following example, `iterate outer` creates a branch from the inner DO loop to the point just before the outer ENDDO, associated with the labeled DO loop, while the `iterate` without a label creates a branch to just before the ENDDO of the inner DO loop:

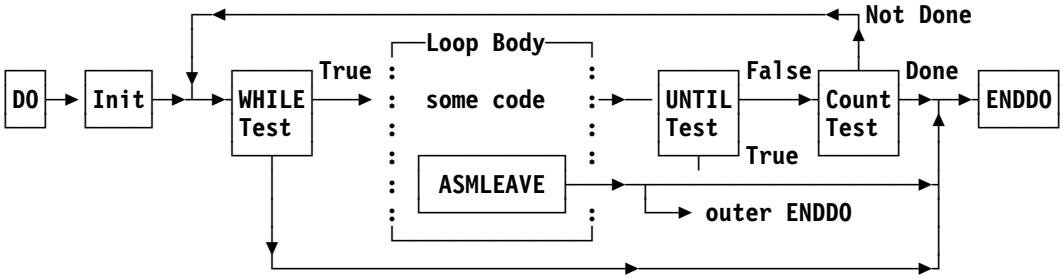
```
outer  do while=2
        do while=4
          mvc a,d
          if (clc,a,eq,b)
             iterate outer
          else
             iterate
          endif
        enddo
      enddo
```

This generates the following:

```
 outer    do while=2
+outer    DC    0H
+                  BC            15,#@LB2
+#@LB3                 DC 0H
           do while=4
+                  BC            15,#@LB7
+#@LB8                 DC 0H
             mvc a,d
             if (clc,a,eq,b)
+                clc             a,b
+           BC           15-8,#@LB11
                iterate outer
+           BC           15,#@LB2
             else
+             BC             15,#@LB13
+#@LB11                DC  0H
                iterate
+         BC             15,#@LB7
             endif
+#@LB13                DC  0H
           enddo
+#@LB7           DC             0H
+                BC             4,#@LB8
         enddo
+#@LB2           DC             0H
+                BC             2,#@LB3
+#@LB1                 DC                    0H
```

Leaving a Nested DO

It is possible to leave a nested DO macro by specifying a label on the DO macro and the same label as a parameter on a contained ASMLEAVE macro. Here is the flowchart for this structure:



If a label is not specified, then the current macro is exited.

In the following example, `asmleave loop` breaks from the inner DO loop to the end of the outer (labeled) DO loop, while the `asmleave` without a label just breaks to the end of the current DO loop:

```
loop   do while=2
        do while=4
          mvc a,d
          if (clc,a,eq,b)
             asmleave loop
          else
             asmleave
          endif
        enddo
       enddo
```

This generates the following:

```
                    loop do while=2
                   +loop   DC    0H
                   +               BC            15,#@LB2
                   +#@LB3              DC 0H
                              do while=4
                   +               BC            15,#@LB7
                   +#@LB8              DC 0H
                                 mvc a,d
                                 if (clc,a,eq,b)
                   +               clc             a,b
                   +         BC           15-8,#@LB11
                                  asmleave loop
                   +         BC           15,#@LB1
                                 else
                   +           BC             15,#@LB13
                   +#@LB11               DC  0H
                                  asmleave
                   +       BC             15,#@LB6
                                endif
                   +#@LB13               DC  0H
                              enddo
                   +#@LB7          DC             0H
                   +               BC             4,#@LB8
                   +#@LB6               DC                    0H
                            enddo
                   +#@LB2          DC             0H
                   +               BC             2,#@LB3
                   +#@LB1               DC                    0H
```

Explicit Specification

If you want to specify an explicit BXH or BXLE loop terminator, you may do so by including it in the form of a positional parameter:

```
DO  BXH,FROM=(Rx,i),TO=(Ry+1,j),BY=(Ry,k)
  Code for F
ENDDO
```

generating, for example:

```
DO  BXH,FROM=(R1,20),TO=(R3,100),BY=(R2,4)
        LA  R1,20
        LA  R3,100
        LA  R2,4
#@LB2                 DC  0H
*         Code for F
ENDDO
#@LB3           DC             0H
                BXH R1,R2,#@LB2
```

The FROM and either the BY or TO keywords must be present in order to provide register designations required for the generation of the BXH or BXLE instruction. The register specified for the BY keyword is used unless it is not present, in which case the one for the TO keyword is used.

Counting

This case applies when a count is to be decremented by 1 each time, and the loop is to be terminated when the count reaches zero. This is achieved by specifying just the FROM keyword. In the situation where only two parameters are used, a BCT loop terminator is generated.

For example:

```
         DO  FROM=(Rx,number)
           Code for F
         ENDDO
```

generates:

```
         DO  FROM=(R15,3)
                 LA  R15,3
#@LB2                  DC  0H
           Code for F
         ENDDO
#@LB3           DC              0H
                BCT             R15,#@LB2
```

For a slightly shorter loop, write the FROM keyword with three parameters to designate an additional register. In this case, a BCTR is generated as the loop terminator.

For example:

```
         DO  FROM=(Rx,=A(TEST),Ry)
           Code for F
         ENDDO
```

generates:

```
         DO  FROM=(R15,=A(LIMIT),R14)
           LA   R14,#@LB2
                   L   R15,=A(LIMIT)
#@LB2                  DC  0H
           Code for F
         ENDDO
#@LB3           DC              0H
                BCTR            R15,R14
```

If no value appears in the FROM keyword, the load instruction is not generated.

Backward Indexing

To index backward through an array (from high to low storage addresses), you need a BXH test, to end the loop when the lowest address is reached. This may be achieved in two ways.

The first way uses all three keywords, with numeric values for the FROM and TO values i and j, where the FROM value i is greater than the TO value j. Although no test on the BY value k is performed, it should be negative. Also, while the FROM and TO values i and j need not be positive, they are assumed to be negative numerics if and only if a leading minus sign occurs.

Thus, with i greater than j:

```
         DO  FROM=(Rx,6),TO=(Ry+1,-6),BY=(Ry,-4)
           Code for F
         ENDDO
```

generates:

```
         DO  FROM=(R1,6),TO=(R3,-6),BY=(R2,-4)
                 LA  R1,6
             LH  R3,=H'-6'
             LH  R2,=H'-4'
#@LB2                DC  0H
           Code for F
         ENDDO
#@LB3          DC              0H
               BXH R1,R2,#@LB2
```

The other way is to omit the TO keyword. The BY value k is a negative numeric (it has a leading minus sign), indicating backward indexing. Although no test on the register number Ry is performed, it must have an odd value.

When k is negative, then:

```
         DO  FROM=(Rx,=A(END-START)),BY=(Ry,-2)
           Code for F
         ENDDO
```

generates:

```
         DO  FROM=(R1,=A(END-START)),BY=(R3,-2)
                   L   R1,=A(END-START)
             LH  R3,=H'-2'
#@LB2                DC  0H
           Code for F
         ENDDO
#@LB3          DC              0H
               BXH R1,R3,#@LB2
```

Forward Indexing

To index forward through an array (from low to high storage addresses), you need a BXLE test, to end the loop when the highest address is reached. If no explicit terminator is specified, and if none of the preceding combinations of keywords and values exist, then forward indexing is assumed, and a BXLE terminator is generated.

For example:

```
         DO  FROM=(Rx,1),TO=(Ry+1,10),BY=(Ry,2)
           Code for F
         ENDDO
```

generates:

```
         DO  FROM=(R1,1),TO=(R3,10),BY=(R2,2)
                LA  R1,1
                LA  R3,10
                LA  R2,2
#@LB2                 DC  0H
           Code for F
         ENDDO
#@LB3           DC              0H
                BXLE R1,R2,#@LB2
```

Register Initialization

If you wish to load a register yourself, or the register remains loaded from a previous operation, then omitting the corresponding value field prevents generation of a register load instruction. If you supplied one or more of the values i, j, or k, thus indicating that you want the macro processor to generate the L, LH, LR, or LA instructions, the following rules apply.

For a positive number greater than zero and less than 4096, an LA is generated. The L or LH instruction is generated when a value is identified as a negative number (determined by the presence of a leading minus sign), or a positive number greater than 4095. The value is also converted to a literal, thus generating `=F'number'` or `=H'number'` as appropriate, and is substituted as the second operand of the load instruction.

If any value is zero, an SR to clear the designated register is generated.

In all other cases (non-numeric or undefined values, as indicated by the type attribute of the macro), an L instruction is generated. In this case, whatever is present for the value is directly substituted as the second operand of the instruction. If it is a literal, it is your responsibility to supply the equal sign.

The table below summarizes the rules followed in initializing registers.

Value given	Instruction generated
None	None
Zero	`SR Rx,Rx`
0 <= number < 4096	`LA Rx,number`
-32767 <= number < 0, or 4096 < number < 32768	`LH Rx,=H'number'`
number < -32767, or number >= 32768	`L Rx,=F'number'`
(Register number)	`LR Rx,Register number`
Other	`Rx,Other`

Table 5. Generated Instructions for Given Values

The UNTIL and WHILE Keywords

The test generated by the UNTIL keyword, as with those generated by the indexing group, is used at the loop termination. The test generated by the WHILE keyword, on the other hand, tests whether to enter a loop at all prior to its execution. For both keywords, the parameterization is identical to that of the IF macro. The UNTIL and WHILE operands accept compound predicates in the same format as used on the IF statement, with the exception that the CC= keyword operand is not allowed.

The DO WHILE example:

```
         DO  WHILE=(TM,FLAGS,X'80',O)
           Code for F
         ENDDO
```

generates:

```
         DO  WHILE=(TM,FLAGS,X'80',O)
                BC            15,#@LB2
#@LB3               DC 0H
           Code for F
         ENDDO
#@LB2          DC              0H
               TM              FLAGS,X'80'
               BC              1,#@LB3

         BC  1,L1
```

The DO UNTIL is coded in the same manner:

```
         DO  UNTIL=(TM,FLAGS,X'80',O)
           Code for F
         ENDDO
```

and generates:

```
         DO  UNTIL=(TM,FLAGS,X'80',O)
#@LB2                DC  0H
           Code for F
         ENDDO
#@LB3          DC              0H
               TM              FLAGS,X'80'
               BC              15-1,#@LB2
```

It is possible to create a compound DO with both UNTIL and WHILE parameters on the same macro. For example:

```
         DO  WHILE=(SRP,AMOUNT,64-3,5,M),UNTIL=10
           Code for F
         ENDDO
```

generates:

```
         DO  WHILE=(SRP,AMOUNT,64-3,5,M),UNTIL=10
#@LB2               DC 0H
              SRP AMOUNT,64-3,5
               BC              15-4,#@LB1
           Code for F
         ENDDO
#@LB5          DC              0H
               BC              15-10,#@LB2
#@LB1                DC                    0H
```

The operand formats for the WHILE and UNTIL keywords are the same as those of the IF-type macros and can be used with Boolean operators as in the following example:

```
         DO WHILE=(CLI,WORD1,EQ,2,OR,CLI,WORD1,EQ,4),                  X
               UNTIL=(CLI,WORD1,EQ,1,OR,CLI,WORD1,EQ,3)
            Code for F
         ENDDO
```

generates (with ASMMREL ON in effect):

```
         DO WHILE=(CLI,WORD1,EQ,2,OR,CLI,WORD1,EQ,4),                  X
               UNTIL=(CLI,WORD1,EQ,1,OR,CLI,WORD1,EQ,3)
#@LB2           DC 0H
                CLI             WORD1,2
                BRC             8,#@LB4
                CLI             WORD1,4
                BRC             15-8,#@LB1
#@LB4           DC              0H
            Code for F
         ENDDO
#@LB5           DC              0H
                CLI             WORD1,1
                BRC             8,#@LB7
                CLI             WORD1,3
                BRC             15-8,#@LB2
#@LB7           DC              0H
#@LB1             DC                    0H
```

To obtain the equivalent capability of logical expressions for looping operations, the DOEXIT or EXITIF macro may also be used, within their respective sets. For a Boolean WHILE, the above macros are placed immediately following the DO or STRTSRCH while for the UNTIL the placement of these macros is immediately before the ENDDO or ENDLOOP.

EXITIF can only be coded within a STRTSRCH structure. Multiple DOEXITs are allowed.

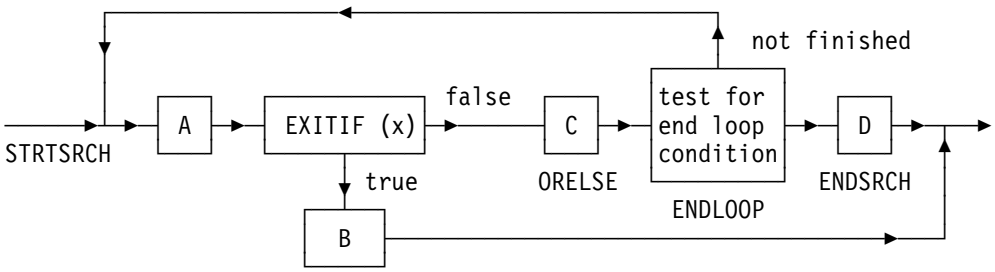
DOEXIT must be placed within a DO macro set, and can be placed within a nested IF macro set. In the following example the DOEXIT macro statement causes the generation of a branch instruction to a label at the ENDDO macro statement.

```
do from=2
   if (clc,a,eq,b)
    mvc a,d
    doexit (2)
   else
     mvc g,h
   endif
enddo
```

The SEARCH Macro Set

The SEARCH macro set is provided to allow more complex loops to be coded. All of the operands noted above for DO loops may also be used on the STRTSRCH macro.

The flowchart for the SEARCH macro set is:



The general structure of the SEARCH macro set is:

```
STRTSRCH  (any DO-type loop operands)
  Process Code A
EXITIF    (any IF-type operands)
  Process Code B
ORELSE
  Process Code C
ENDLOOP
  Process Code D
ENDSRCH
```

Multiple EXITIFs are permissible. However, for each EXITIF, an ORELSE must appear at some point in the code before the next EXITIF. However, the last ORELSE (the one before the ENDLOOP macro) is optional.

For example:

```
   STRTSRCH  UNTIL=(TM,0(R4),X'55',NO),WHILE=(CH,R9,LT,=H'58')
     Process A
   EXITIF CC=8
     Process B
   ORELSE
     Process C
   ENDLOOP
     Process D
   ENDSRCH
```

generates:

```
         STRTSRCH UNTIL=(TM,0(R4),X'55',NO),WHILE=(CH,R9,LT,=H'58')
#@LB3                DC 0H
               CH              R9,=H'58'
               BC              15-4,#@LB2
           Process A
         EXITIF CC=8
           BC           15-8,#@LB9
           Process B
         ORELSE
             BC              15,#@LB1
#@LB9                  DC  0H
           Process C
         ENDLOOP
#@LB6          DC              0H
               TM              0(R4),X'55'
               BC              15-14,#@LB3
#@LB2                DC                      0H
           Process D
         ENDSRCH
#@LB1                 DC  0H
```

Another example:

```
            STRTSRCH  WHILE=(CLM,R2,M1,GE,D2(B2)),UNTIL=P
*           No Process A
            EXITIF  Z,AND,                                          X
              LTR,R2,R3,O,ORIF,                                     X
              (CLC,DEC(L,B),EQ,=C'WORD'),AND,                       X
              NP
              Process B
            ORELSE
              Process C
            EXITIF  CC=5
              Process D
            ENDLOOP
              Process E
            ENDSRCH
```

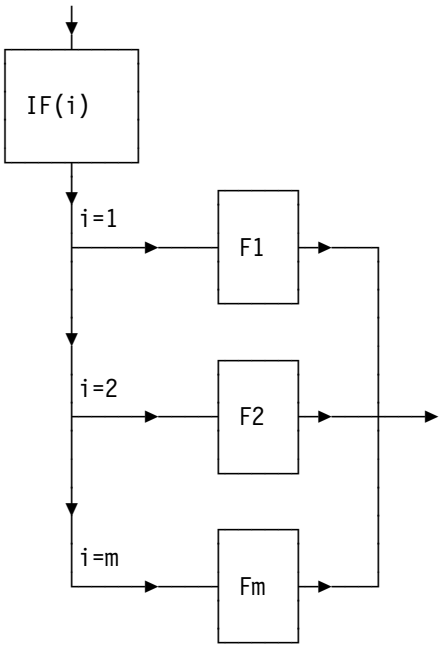
generates:

```
         STRTSRCH  WHILE=(CLM,R2,M1,GE,D2(B2)),UNTIL=P
#@LB3               DC 0H
                CLM R2,M1,D2(B2)
                BC              15-11,#@LB2
*        No Process A
         EXITIF  Z,AND,                                             X
               LTR,R2,R3,O,ORIF,                                    X
               (CLC,DEC(L,B),EQ,=C'WORD'),AND,                      X
               NP
                BC              15-8,#@LB9
                LTR             R2,R3
                BC              1,#@LB10
#@LB9           DC              0H
                CLC             DEC(L,B),=C'WORD'
                BC              15-8,#@LB11
           BC           15-13,#@LB11
#@LB10                 DC  0H
           Process B
         ORELSE
             BC              15,#@LB1
#@LB11                 DC  0H
           Process C
         EXITIF  CC=5
           BC           15-5,#@LB12
           Process D
         ENDLOOP
           BC              15,#@LB1
#@LB12                 DC  0H
#@LB6           DC              0H
                BC              15-2,#@LB3
#@LB2                DC                     0H
           Process E
         ENDSRCH
#@LB1                  DC  0H
```

The CASE Macro Set

The CASE macro set selects one of a set of functions for execution, depending on the value of an integer found in a specified register. The determination of which of the functions is to be executed involves the use of either an address vector (sequence of addresses) or a branch vector (sequence of branch instructions).

The flowchart for the CASE program figure is:



The macro is written like this:

```
CASENTRY  Rx,POWER=n,VECTOR=listtype
  CASE  a,d
    Code for F1
  CASE  b,c
    Code for F2
  CASE  f
    Code for F3
    .
    .
    .
  CASE  t
    Code for Fm
ENDCASE
```

Notes:

1. *listtype* can be either B or BR.
2. Statements between the CASENTRY macro and the first CASE statement are assembled, but not executed. Statements should not be placed between CASENTRY and CASE.
3. An integer cannot be used more than once in a CASENTRY structure.

Where the case numbers a, b, ..., t are either members of a set of integers greater than zero, or nonzero multiples of a power of 2 (for example, 4, 12, and 16). Zero

(0) is not a valid case number. Rx is a positional operand that specifies a general register containing the case number. The keyword operands POWER and VECTOR are optional.

The operand POWER=n (where n is an integer) refers to a power of 2 and indicates that the case numbers are multiples of that power of 2. Thus, POWER=3 indicates that the case numbers are multiples of 8.

The default value for POWER is 0 which indicates that the case numbers are positive integers that are necessarily powers of 2.

The operand VECTOR=B or VECTOR=BR indicates that a branch vector is to be generated rather than an address vector. Fewer instructions are generated for branch vectors. However, you must be sure that the branch vector table is addressable by the initialization code, that the code for each of the cases is addressable, and that the code after the ENDCASE macro is addressable by a current base register.

Register 0 may not be used as the case value register (Rx).

It is your responsibility to load the desired case number into Rx and to ensure that it is within the indicated range. The macro expansion then adjusts this value according to the POWER value (whether explicitly or implicitly specified), so that the correct CASE is selected. The content of the register indicated in the CASENTRY statement is destroyed and is only required during the execution of the initial code generated by the macro expansion. Hence, it is possible to use the same register for other purposes within the function code for any CASE(s).

This example of a CASE macro uses case numbers 1, 2, 3, 4, and 5:

```
CASENTRY  Rx
  CASE 2,1,4
    Code for F1
  CASE 5
    Code for F2
ENDCASE
```

This is interpreted to mean that if a 1, 2, or 4 is present in general register Rx, the code for F1 is executed. If a 5 is present, the code for F2 is executed. If a value of 3 is in Rx, no function code is to be executed. In all cases, control is then to be passed to the code after the ENDCASE macro.

The example generates:

```
         SLA  Rx,2
         A    Rx,L3
         L    Rx,0(Rx)
         BCR  15,Rx
      L3 DC   A(L1)
      L4 DC   0H
           Code for F1
         L    Rx,L1
         BCR  15,Rx
      L5 DC   0H
           Code for F2
         L    Rx,L1
         BCR  15,Rx
      L1 DC   A(L2)
         DC   A(L4)
         DC   A(L4)
         DC   A(L2)
         DC   A(L4)
         DC   A(L5)
      L2 DC   0H
```

This example shows a CASE macro using a branch vector and case number that are multiples of 8:

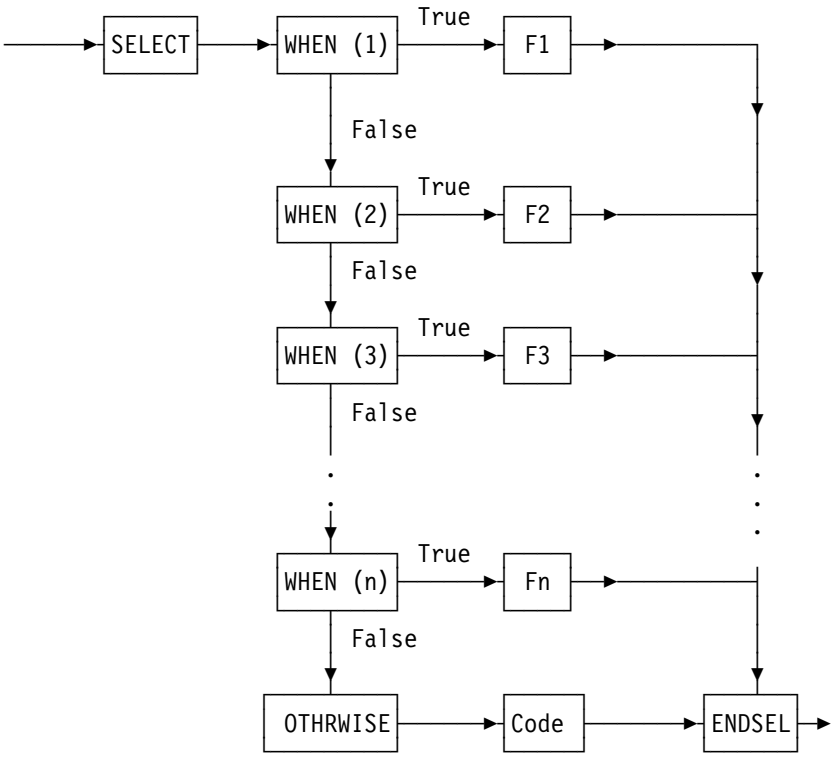
```
         CASENTRY  Rx,POWER=3,VECTOR=B
           CASE 8,24
             Code for F1
           CASE  16,32
             Code for F2
         ENDCASE
```

The example generates:

```
         CASENTRY  Rx
               SLA Rx,2-0
             A    Rx,#@LB3
             L    Rx,0(,Rx)
             BCR  15,Rx

#@LB3                DC A(#@LB1)
           CASE 2,1,4
#@LB4                   DC  0H
             Code for F1
           CASE 5
             L     Rx,#@LB1
             BCR   15,Rx
#@LB5                   DC  0H
             Code for F2
         ENDCASE
             L     Rx,#@LB1
             BCR   15,Rx
#@LB1                   DC A(#@LB2)
                 DC    A(#@LB4)
                 DC    A(#@LB4)
                 DC    A(#@LB2)
                 DC    A(#@LB4)
                 DC    A(#@LB5)
#@LB2                   DC  0H
```

The SELECT Macro Set

The SELECT macro set selects one of a set of functions for execution, depending on the result of a comparison. The flowchart for the SELECT program figure is:



OTHRWISE is optional.

This example uses the SELECT, WHEN, OTHRWISE, and ENDSEL macros:

```
SELECT CLI,0(R6),EQ          Defines the comparison
  WHEN  (X'20')
    Code for F1
  WHEN  (1,5,13)
    Code for F2
  WHEN  (3,7,15)
    Code for F3
  OTHRWISE
    Code for F4
ENDSEL
```

It generates:

```
         SELECT CLI,0(R6),EQ           Defines the comparison
           WHEN  (X'20')
         CLI                    0(R6),X'20'
         BC           15-8,#@LB2
             Code for F1
           WHEN  (1,5,13)
         BC              15,#@LB1               SKIP TO END
#@LB2                    DC  0H
         CLI                    0(R6),1
         BC               8,#@LB5
         CLI                    0(R6),5
         BC               8,#@LB5
         CLI                    0(R6),13
         BC           15-8,#@LB4
#@LB5                    DC  0H
             Code for F2
           WHEN  (3,7,15)
         BC              15,#@LB1               SKIP TO END
#@LB4                    DC  0H
         CLI                    0(R6),3
         BC               8,#@LB7
         CLI                    0(R6),7
         BC               8,#@LB7
         CLI                    0(R6),15
         BC           15-8,#@LB6
#@LB7                    DC  0H
             Code for F3
           OTHRWISE
         BC              15,#@LB1               SKIP TO END
#@LB6                    DC  0H
             Code for F4
         ENDSEL
#@LB1                    DC  0H
```

Here is another example of the SELECT Macro Set:

```
SELECT CLM,2,B'1100',EQ
WHEN (=C'AA',=C'BB')
  Process A
WHEN =C'AB'
  Process B
WHEN =C'12'
  Process C
ENDSEL
```

It generates:

```
        CLM   2,B'1100',=C'AA'
        BC    8,LB3
        CLM   2,B'1100',=C'BB'
        BC    15-8,LB2
LB3     DC    0H
          Process A
        B     LB1
LB2     DC    0H
        CLM   2,B'1100',=C'AB'
        BC    15-8,LB4
          Process B
        B     LB1
LB4     DC    0H
        CLM   2,B'1100',=C'12'
        BC    15-8,LB6
          Process C
LB6     DC    0H
LB1     DC    0H
```

The SELECT group allows a SELECT with no operands followed by WHEN macros with IF style operands. This produces the same structure as the IF/ELSEIF/ELSE/ENDIF macros.

For example: :

```
        SELECT
        WHEN (CLI,WORD1,EQ,1),OR,(CLI,WORD1,EQ,2),OR,(CLI,WORD1,EQ,3)
           <code for first condition>
        WHEN (CLI,WORD2,EQ,2),AND,(CLI,WORD3,EQ,3)
           <code for second condition>
        OTHRWISE
           <otherwise  code>
        ENDSEL
```

generates (assuming that ASMMREL ON has been coded earlier):

```
         SELECT
         WHEN (CLI,WORD1,EQ,1),OR,(CLI,WORD1,EQ,2),OR,(CLI,WORD1,EQ,3)
               CLI             WORD1,1
               BRC             8,#@LB3
               CLI             WORD1,2
               BRC             8,#@LB3
               CLI             WORD1,3
           BRC          15-8,#@LB2
#@LB3                  DC  0H
            <code for first condition>
         WHEN (CLI,WORD2,EQ,2),AND,(CLI,WORD3,EQ,3)
         BRC             15,#@LB1               SKIP TO END
#@LB2                  DC  0H
               CLI             WORD2,2
               BRC             15-8,#@LB4
               CLI             WORD3,3
           BRC          15-8,#@LB4
            <code for second condition>
         OTHRWISE
         BRC             15,#@LB1               SKIP TO END
#@LB4                  DC  0H
            <otherwise  code>
         ENDSEL
#@LB1                  DC  0H
```

Chapter 3. Using the Disassembler

Introduction to the Disassembler 41
Invoking the Disassembler 42
Invoking the Disassembler on MVS 42
MVS JCL Example 42
Disassembler Options on MVS 44
Invoking the Disassembler on CMS 44
CMS Example 45
Disassembler Options on CMS 46
Invoking the Disassembler on VSE 46
VSE JCL Example: 47
Disassembler Options on VSE 47
Control Statements 48
Module-CSECT Statement (required) 48
Format 49
DATA-only Statement (optional) 49
INSTR-only Statement (optional) 49
DS-area Statement (optional) 50
DSECT Definitions (optional) 50
ULABL Statements 51
USING Statements 51
COPY Statement (optional) 52
Comment Statement (optional) 52
Disassembling a Module for the First Time 52
Output Description 53
SYSPUNCH (SYSPCH for VSE) Content 53
SYSPRINT (SYSLST for VSE) Content 54
Disassembler CMS Messages 55
Disassembler Messages 57

Introduction to the Disassembler

The Disassembler produces assembler language source statements and a pseudo-listing using object code as input. You can use the Assembler Language source file and listing for purposes such as program understanding, debugging, and recovery of lost source code.

ASMDASM is a two-pass disassembler which produces an assembler language source program from a CSECT within any of the following:

MVS An object module, a program object, or a load module.

CMS An object deck, or a CMS Module.

VSE An object module, or a phase.

Control statements permit specification of areas containing instructions or data or uninitialized data areas, provide base registers so that symbolic labels are created during disassembly, and define the DSECTs used during disassembly.

Registers are denoted thus:

- Access Registers are denoted by A0, A1,...A15.

© Copyright IBM Corp. 1992, 2004

- Control Registers are denoted by C0, C1,...C15.
- Floating Point Registers are denoted by F0, F1,...F15.
- General Purpose Registers are denoted by R0, R1,...R15.
- Vector Registers are denoted by V0, V1,...V15.

The Disassembler provides informational comments for recognized SVCs, and for various branch instructions to aid in creating a documented source program.

A Warning about Copyright

When you use this utility you must be aware of and respect the intellectual property rights of others. You are not authorized to use this utility to disassemble, copy or create assembly listings or disassembled Assembler Language source code in violation of any contractual or other legal obligation. You are authorized to use this utility only for object code for which you have verified you have the right to perform disassembly.

The Disassembler normally scans the object code for the strings specified below. If any one of these is found, then the Disassembler issues message ASMD010 and the disassembly stops.

The Disassembler searches for the following data:

- (c)
- (C)
- © at code point X'B4'
- "Copyright" in any combination of upper case and lower case letters.

Invoking the Disassembler

On MVS, the Disassembler processes a program object, a load module, or an object module.

On CMS, the Disassembler processes either a CMS Module, or an object deck.

On VSE, the Disassembler processes either an object module or a phase.

For details on the resulting output for each operating system see “Output Description” on page 53.

Invoking the Disassembler on MVS

On MVS you invoke the Disassembler as a batch program using Job Control Language (JCL). The following sections describe the job control language statements you can use.

MVS JCL Example

```
//DISASM   EXEC PGM=ASMDASM,PARM='options'                          1
//SYSLIB   DD DSN=user.loadlib,DISP=SHR                             2
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=3630                             3
//SYSPUNCH DD DSN=user.command.asm,DISP=(,CATLG),                   4
//            UNIT=SYSDA,DCB=BLKSIZE=3200,
//            SPACE=(TRK,(5,2),RLSE)
//SYSIN DD *
module_name csect_name                                              5
other control statements

⋮
//*COPYLIB DD DSN=user.copy.name,DISP=SHR                           6
```

Figure 6. Sample Disassembler MVS JCL

The following details explain the lines of JCL in Figure 6 highlighted with a number (such as **1**).

1 Replace *options* with any Disassembler options that you want to use. For a list of options, see "Disassembler Options on MVS" on page 44.

EXEC PGM=ASMDASM runs the Disassembler program named ASMDASM.

2 To disassemble an object module, replace *user.loadlib* with either:

- The name of the sequential data set containing the object module
- The name of the PDS, followed by the name of the member in parentheses, containing the object module
- The name of the PDSE, followed by the name of the member in parentheses, containing the program object. (DFSMS/MVS 1.3 or higher is required to support this).

To disassemble a load module, replace *user.loadlib* with the name of the PDS containing the load module. Specify the load module *pds_member* in the Module-CSECT process statement (for details, see **5**).

To disassemble a program object, replace *user.loadlib* with the name of the PDSE containing the program object. Specify the program object *pdse_member* in the Module-CSECT process statement (for details, see **5**).

SYSLIB specifies the object module or, in combination with the *pds_member* or *pdse_member* in **5**, the load module or program object to be disassembled.

3 If you require the assembly listing in a file, enter the name of the data set.

The optional **SYSPRINT DD** statement specifies the output file for the assembly listing. You must specify the BLKSIZE as a multiple of 121. RECFM=FBA,LRECL=121 is hard-coded.

4 If you require the output disassembled source in a file, enter the name of the data set.

The **SYSPUNCH DD** statement is an optional statement which specifies an output file for the disassembled source. You must specify the BLKSIZE as a multiple of 80. RECFM=FB,LRECL=80 is hard-coded.

5 Replace the Module-CSECT control statement, consisting of *module_name* and *csect_name*, with appropriate values for the SYSLIB you specified in **2**. If you specified an object module or a program object in **2**, then *pds_member* or *pdse_member* is ignored. For details, see the "Module-CSECT Statement (required)" on page 48.

Add any other control statements below the Module-CSECT statement.

SYSIN DD contains the control statements. You must specify the module-CSECT statement. You must specify a BLKSIZE in a multiple of 80. RECFM=FB,LRECL=80 is hard-coded.

6 If you use the COPY control statement enter the COPYLIB DD statement.

The **COPYLIB DD** statement contains control statement members selected by COPY control statement. The Disassembler opens this file only if you use the COPY control statement. You must specify a BLKSIZE in a multiple of 80. RECFM=FB,LRECL=80 is hard-coded.

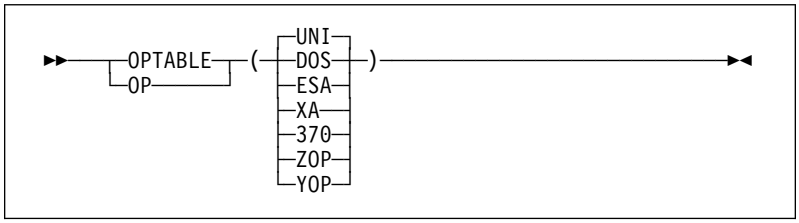
Disassembler Options on MVS

On MVS you can specify the following options in the PARM field:

COPYRIGHTOK Allow disassembly of copyrighted module. If you use the COPYRIGHTOK option then message ASMD008 is printed at the start of the listing.

HEX Generate the offset in machine instructions as a hexadecimal value.

OPTABLE Specifies the operation code table to be used in disassembling CSECTs.



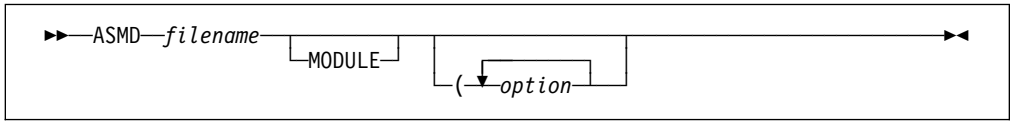
NEWNUM Allow any numeric field within a control statement to be specified either as a decimal value (a sequence of decimal digits) or a hexadecimal value (enclosed in single quotation marks and preceded by the letter X).

VSESVC Use the VSE description for SVCs, **not** the MVS description. Use this option when disassembling VSE code while running on MVS.

You can specify any of the above options together in the PARM string in any order, separated by a comma or space.

Invoking the Disassembler on CMS

On CMS you invoke the Disassembler with the **ASMD** command.



The ASMD command allocates all required files and then passes control to the Disassembler module, ASMDASM. If you enter any file definitions before the ASMD command is issued, then they are used.

The **MODULE** keyword is required if and only if you are disassembling a CMS module.

The file definitions are described below:

SYSIN This file contains control statements of which the module-CSECT statement is required.

The ASMD command issues the following FILEDEF command:

```
FILEDEF SYSIN DISK filename DISASM * (RECFM FB LRECL 80 BLOCK 16000
```

SYSLIB Specifies the file name of the module to be disassembled.

For object decks, the ASMD command issues the following FILEDEF command:

```
FILEDEF SYSLIB DISK filename TEXT * (RECFM FB LRECL 80 BLOCK 16000
```

For CMS Modules, no FILEDEF command is issued.

SYSPRINT Specifies the output file for the disassembler listing

The ASMD command issues the following FILEDEF command:

```
FILEDEF SYSPRINT DISK filename LISTING * (RECFM FBA LRECL 121 BLOCK 1210
```

SYSPUNCH Specifies the output file for the disassembler source.

The ASMD command issues the following FILEDEF command:

```
FILEDEF SYSPUNCH DISK filename PUNCH * (RECFM FB LRECL 80 BLOCK 16000
```

COPYLIB This contains control statement members selected by COPY control statement. Opened only if you use the COPY control statement.

The ASMD command issues the following FILEDEF command:

```
FILEDEF COPYLIB DISK CMSLIB MACLIB * (RECFM FB LRECL 80 BLOCK 8000
```

Before using the COPY control statement, you must issue the CMS command GLOBAL MACLIB to identify the MACLIBs to be searched for control statement members. For more information on the GLOBAL MACLIB command, see the applicable *CMS Command and Macro Reference*.

CMS Example

This example disassembles the CSECT COMMAND in object file PROCESS TEXT.

The following command identifies PROCESS TEXT as the object file:

```
ASMD PROCESS
```

For this example, the predefined control statements file PROCESS DISASM contains only a Module-CSECT statement:

```
ANYNAME COMMAND
```

where ANYNAME is ignored, and COMMAND identifies the CSECT to be disassembled.

The Disassembler outputs a listing in the file PROCESS LISTING, and the disassembled source in the file PROCESS PUNCH.

Disassembler Options on CMS

On CMS you can specify the following options:

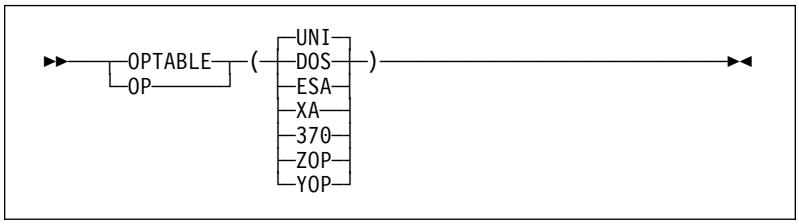
COPYRIGHTOK Allow disassembly of copyrighted module. If you use the COPYRIGHTOK option then the Disassembler prints message ASMD008 at the start of the listing.

DISK Output the LISTING file to disk, this is the default.

ERASE Specifies that the existing files with a file name the same as the file name on the ASMD command, and a file type of LISTING and PUNCH are deleted before the Disassembly is run. Only files on the disk on which the Disassembler writes the new listing and source files are deleted. ERASE is the default.

HEX Generate the offset in machine instructions as a hexadecimal value.

OPTABLE Specifies the operation code table to be used in disassembling CSECTs.



NEWNUM Allow any numeric field within a control statement to be specified either as a decimal value (a sequence of decimal digits) or a hexadecimal value (enclosed in single quotation marks and preceded by the letter X).

NOERASE Do not erase the existing LISTING and PUNCH files before the disassembly is run.

NOPRINT Suppress the writing of the LISTING file.

PRINT Outputs the LISTING file to the virtual printer. The listing is not written to disk.

VSESVC Use the VSE description for SVCs, **not** the MVS description. Use this option when disassembling VSE code while running on CMS.

You can specify any of the above options together in the PARM string in any order, separated by a comma or space.

Invoking the Disassembler on VSE

On VSE you invoke the Disassembler as a batch program using Job Control Language (JCL). The following section describes the job control language statements that you need to run the Disassembler.

VSE JCL Example:

```
// LIBDEF *,SEARCH=(user.library,hlasm.library)                    1
// EXEC ASMDASM,PARM='options'                                     2
module_name csect_name
/*
```

Figure 7. Sample Disassembler VSE JCL

The following details explain the lines of JCL in Figure 7 highlighted with a number (such as **1**).

1 Replace *user.library* and *hlasm.library* with the search chain for the phase or object.

2 Replace *options* with any Disassembler options that you want to use. For a list of options, see "Disassembler Options on VSE." If you do not need any options omit the PARM field.

EXEC ASMDASM runs the Disassembler program named ASMDASM. The Disassembler control statements can follow the EXEC statement, in SYSIPT as in the example shown above. Enter each statement on a separate line, with the last statement followed by the SYSRDR termination control characters `/*` on the last line. Or you can assign SYSIPT to a file.

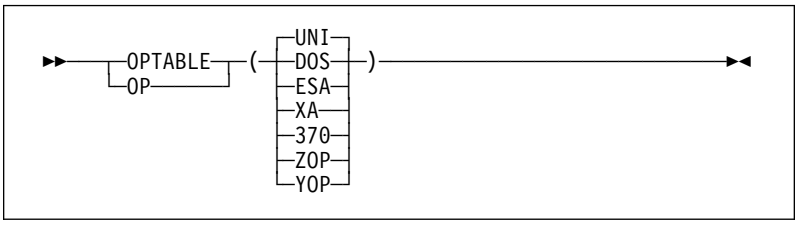
Disassembler Options on VSE

On VSE you can specify the following options:

COPYRIGHTOK Allow disassembly of copyrighted module. If you use the COPYRIGHTOK option the Disassembler prints the message ASMD008 at the start of the listing.

HEX Generate the offset in machine instructions as a hexadecimal value.

OPTABLE Specifies the operation code table to be used in disassembling CSECTs.



MVSSVC Use the MVS description for SVCs, not the VSE description. Use this option when disassembling MVS code while running on VSE.

NEWNUM Allow any numeric field within a control statement to be specified either as a decimal value (a sequence of decimal digits) or a hexadecimal value (enclosed in single quotation marks and preceded by the letter X).

PHASE If the module processed is a PHASE that may also exist as an object then you must specify PHASE.

Note: If you have not specified `PARM='PHASE'` the Disassembler

searches for an object first, and if the object is not found it searches for a phase.

You can specify any of the above options together in the PARM string in any order, separated by a comma or space.

Control Statements

You enter control statements:

MVS In SYSIN and, optionally, in a PDS member specified by a COPY control statement. This member must belong to the PDS specified by the COPYLIB DD statement.

CMS In the file *filename* DISASM (where you invoked the Disassembler using the command ASMD *filename*) and, optionally, in a MACLIB member specified by a COPY control statement. This member must belong to a MACLIB specified by the GLOBAL MACLIB CMS command.

VSE In SYSIPT and, optionally, in a Librarian member specified by a COPY control statement. This member must belong to a library specified in the search chain.

The following rules apply to control statements:

- Columns 1–72 can contain only data.
- Columns 73–80 can be used for any desired purpose. In addition, columns beyond the last specified may be used for any purpose.
- Hexadecimal fields may contain only the hexadecimal digits 0–9 and A–F, while decimal fields may contain only digits 0–9.
- You must specify the module-CSECT statement in the first statement in the input stream. For more details on the Module CSECT statement see “Module-CSECT Statement (required).”
- DSECT definitions may not include any other control statement.
- USING statements for DSECTs must be entered after the DSECT definition.
- DATA-only statements and program USING statements may be entered in any order except within DSECT definitions.

The COPY control statement may be used to switch the input stream to a member of the COPYLIB file which contains additional control statements. COPY statements are not allowed in these supplemental control statement members. That is, COPY statements may not cause nesting COPYing. After the COPY member input is exhausted, the original input stream is resumed. This control statement is especially helpful where a large common DSECT is used by multiple CSECTs in a module.

Module-CSECT Statement (required)

Identifies the module and CSECT to be disassembled. Must be the first control statement in the input stream, and specifies the module name, CSECT name, and optionally CLASS name for program objects.

MVS MVS needs the module name. If you specify the name of a PDS or PDSE in the SYSLIB DD statement, then the module name you specify in the

Module-CSECT statement identifies a load module or a program object to be disassembled, which must be a member of that PDS or PDSE. Otherwise, the name is ignored (but still needed); the PDS or PDSE member or sequential data set identified by the SYSLIB DD statement is assumed to be an object module. The CSECT name is optional. If it is specified, the named CSECT must exist in the module or object module. If omitted, the CSECT with ESDID=0001 is disassembled. MVS

CMS On CMS, you must provide a module name but it is not used for object decks. For object decks, only the first CSECT is disassembled; selection of specific CSECTs is not possible. For CMS modules, the CSECT name is optional. If it is specified, the named CSECT must exist in the module. CMS

VSE On VSE, a phase does not contain any information allowing selection of individual CSECTs. The phase is therefore viewed as one CSECT where the CSECT name is determined by the CSECT name on this statement, if present; otherwise it is the module name. VSE

Format

Free-form, with module name beginning in column 1. At least one space must separate module name and CSECT name and, if specified, CLASS name. The names may be separated by any number of spaces.

DATA-only Statement (optional)

This describes areas of the CSECT being disassembled which contain no instructions. Use of this statement eliminates creation of instructions from constant data, or from areas containing values created during program linking. Up to 256 DATA-only statement may be entered. These statements may occur anywhere in the input stream after the module-CSECT statement, but not within a DSECT definition set.

Table 6. DATA-only statement: format

Column	Contents
1–4	• literal 'DATA'
5 onwards	• one or more spaces • offset to beginning of area, in hexadecimal • one or more spaces • offset to end of area (last byte), in hexadecimal

INSTR-only Statement (optional)

This describes areas of the CSECT being disassembled which are instruction areas. This statement allows the bypassing of the following tests which might identify valid instructions as data:

- 4 consecutive identical bytes
- 6 consecutive valid EBCDIC characters
- next instruction valid opcode

The remaining instruction tests remain in effect, and invalid instructions are still generated as hexadecimal data statements. Up to 256 INSTR-only statements can

be entered. These statements can occur anywhere in the input stream after the module-CSECT statement, but not within a DSECT definition set.

Table 7. INSTR-only statement: format

Column	Contents
1–5	• literal 'INSTR'
6 onwards	• one or more spaces • offset to beginning of area, in hexadecimal • one or more spaces • offset to end of area (last byte), in hexadecimal

DS-area Statement (optional)

This describes areas of the CSECT being disassembled which are uninitialized storage areas. These text areas are cleared to binary zeros before the disassembly begins. Use of this statement forces the creation of DS assembly opcodes, eliminating the creation of instructions or data constants. Up to 256 DS-area statements may be entered. These statements may occur anywhere in the input stream after the module-CSECT statement, but not within a DSECT definition set.

Table 8. DS-area statement: format

Column	Contents
1–2	• literal 'DS'
3 onwards	• one or more spaces • offset to beginning of area, in hexadecimal • one or more spaces • offset to end of area (last byte), in hexadecimal

DSECT Definitions (optional)

A DSECT is defined by a header statement followed by up to 9999 field definition statements. No other control statements may be entered within a DSECT definition. Up to 256 DSECT definitions may be entered.

Table 9. DSECT header statement: format

Column	Contents
1–8	• DSECT name
9 onwards	• one or more spaces • literal 'DSECT' • one or more spaces • number of field statements to follow (decimal)

Table 10 (Page 1 of 2). DSECT field statement: format

Column	Contents
1–8	• field name

Table 10 (Page 2 of 2). DSECT field statement: format

Column	Contents
9 onwards	• one or more spaces • offset to start of field (decimal). Maximum offset is 4095. • one or more spaces • length of field in bytes (decimal). Maximum length is 999.

ULABL Statements

These statements define user labels to be placed on statements within the program. If program base registers are set up with USING statements, these are also generated as symbolic operands on instructions.

Table 11. ULABL statements: format

Column	Contents
1–5	• literal 'ULABL'
6 onwards	• one or more spaces • label name • one or more spaces • offset to start of field (hexadecimal) • one or more spaces • length of the named field (decimal). Maximum length is 999.

USING Statements

These statements define base register usage. Up to 256 USING statements may be entered. These statements permit the Disassembler to convert explicit base-displacement addresses to symbolic labels. Labels created within the program is 7 characters long. The first character is 'A', followed by the 6-hex-digit offset to the label. A USING statement must be entered for each DSECT to be used.

Table 12. USING statements: format

Column	Contents
1–5	• literal 'USING'
6 onwards	• one or more spaces • offset of beginning location for USING range in hexadecimal (this is where the USING statement occurs) • one or more spaces • offset of ending location for USING range in hexadecimal (this is where the DROP statement occurs) • one or more spaces • base register to be used (hexadecimal 1-F) • one or more spaces • type, P=program base, D=DSECT base • one or more spaces • initial base register value (if type P) in hexadecimal • DSECT name (if type D)

COPY Statement (optional)

This switches the control statement input stream to the specified source member in a data set or library, appropriate to the platform. This member contains additional control statements which are read and processed until the COPY control statements member is exhausted. These statements may occur anywhere in the input stream after the module-CSECT statement, but not within a control statement member being copied.

Table 13. COPY statement: format

Column	Contents
1–4	• literal 'COPY'
5 onwards	• one or more spaces • COPY member name

Comment Statement (optional)

The comment statement allows you to enter comments in the control statement stream which is printed as part of the entered statement, but ignored thereafter.

Table 14. Comment statement: format

Column	Contents
1	• literal '*'
2–72	• comment text

Disassembling a Module for the First Time

When you first disassemble a module, do not use the SYSPUNCH (SYSPCH for VSE) output, but print the SYSPRINT (SYSLST for VSE) listing. Use the listing to determine which registers are used as program base registers, their initial values, and their ranges. Make up USING statements for these. Find any places where no instructions should be generated (only constants), and make up data-only statements for these ranges. Find any uninitialized data areas (DS areas), and make up DS statements for these ranges. If you can determine any registers that are bases for areas which can be used for DSECTs, determine the range of valid use, and make up DSECT definitions and USING statements for these. Perform a second disassembly, including the above statements, and creating a source program with the SYSPUNCH (SYSPCH for VSE) output.

Output Description

SYSPUNCH (SYSPCH for VSE) Content

This output contains the disassembled source program. Statement names begin in column 1, mnemonics begin in column 10, operands in col 16, and an occasional comment begins in column 44. A sequence number (by tens) is in columns 73–80. Comments are included to show the macro name associated with SVCs, and other statements are flagged to aid in identification of certain operations:

Instruction / Addresses	Comment
BALR 14,15	std linkage
BALR x,0	address set
other BALRs	non-std linkage
BASR 14,15	std linkage
BASR x,0	address set
other BASRs	non-std linkage
BAL 0,xxx and BAL 1,xxx	parm set brch
BAL x,xxx	perform
BAS 0,xxx and BAS 1,xxx	parm set brch
BAS x,xxx	perform
STM instructions	save regs
LM instructions	restore regs
BCR 15,R14	exit
absolute location hexadecimal 10	CVT address
absolute location hexadecimal 4C	CVT address
other absolute locations	PSA reference
EX instructions	run instr opcode
L instructions	reference to ADCONS

When used explicitly in instructions, registers are denoted by:

- Access Registers by A0, A1,...A15.
- Control Registers by C0, C1,...C15.
- Floating Point Registers by F0, F1,...F15.
- General Purpose Registers by R0, R1,...R15.
- Vector Registers by V0, V1,...V15.

An ASMDREG macro is generated at the end of the program to create the appropriate EQU statements for the symbols defining the Access, Control, Floating Point, General Purpose, and Vector registers. If any DSECTs were defined in the SYSIN file, they are near the end of the source program. ASMDREG is installed by default in PRD2.PROD. Check with your systems programmer if HLASM Toolkit was installed in a different sublibrary.

SYSPRINT (SYSLST for VSE) Content

Directory information Contains data from the directory entry of the module containing the CSECT to be disassembled, if available.

ESD table A formatted list of all external symbol entries found in the module.

RLD table A formatted listing of all relocation dictionary entries pertaining to this CSECT.

User entered records A list of the records entered by you, with diagnostics, if appropriate.

Label table A list of all the labels to be used during disassembly including those developed from ESD entries, RLD entries, and generated names resulting from USING statements processing.

Text A storage-dump formatted listing of the text which comprises the CSECT being disassembled.

Source listing A printout of the generated source program statements, including the hexadecimal value which resulted in the instruction's creation.

Note: The number of lines per page assumed in the disassembler listing is 60.

Disassembler CMS Messages

ASMDCMS002E File *fn ft fm* not found

Explanation: The file name you included in the ASMD command does not correspond to the names of any of the files on your disks.

Supplemental Information: The variable file name, file type, and file mode in the text of the message indicate the file that could not be found.

System Action: RC=28. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMD with the correct file name.

ASMDCMS003E Invalid option *option*

Explanation: You have included an incorrect option that is not correct with your ASMD command.

Supplemental Information: The variable option in the text of the message indicates the option that is not correct.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the ASMD command, and reissue the command with the correct option.

ASMDCMS004E Improperly formed option *option*

Explanation: You have included an improperly formed option with your ASMD command.

Supplemental Information: The variable option in the text of the message indicates the improperly formed option.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the ASMD command, and reissue the command with the correct option.

ASMDCMS005E Truncation of options may have occurred because of tokenized PLIST format

Explanation: The options have been passed to the ASMD command in tokenized PLIST format. Any options passed might have been truncated to 8 characters. This message is only issued when an error has been detected in one of the options that was specified.

System Action: The options are accepted as entered but might have been truncated.

Programmer Response: If the options have been truncated, invoke the ASMD command with the extended parameter list.

ASMDCMS006E No read/write disk accessed

Explanation: Your virtual machine configuration does not include a read/write disk for this terminal session, or you failed to specify a read/write disk.

System Action: RC=36. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Issue an ACCESS command specifying a read/write disk.

ASMDCMS007E File '*fn ft fm*' does not contain fixed length 80 character records

Explanation: The control file you specified in the ASMD command does not contain fixed-length records of 80 characters.

Supplemental Information: The variable file name, file type, and file mode in the text of the message indicate the file that is in error.

System Action: RC=32. The command cannot be processed.

Programmer Response: You must reformat your file into the correct record length. CMS XEDIT or COPYFILE can be used to reformat the file.

ASMDCMS010E File name omitted and FILEDEF '*ddname*' is undefined

Explanation: You have not included a file name in the ASMD command, and no FILEDEF could be found for the *ddname* specified.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMD command and specify a file name, or issue a FILEDEF for the *ddname* specified.

ASMDCMS011E File name omitted and FILEDEF '*ddname*' is not for DISK.

Explanation: You have not included a file name in the ASMD command, and the FILEDEF for the *ddname* specified is not for DISK.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the ASMD command and specify a file name, or reissue the FILEDEF for the ddname specified with a device type of 'DISK'.

ASMDCMS038E File name conflict for the SYSIN FILEDEF.

Explanation: The file name specified on the ASMD command conflicts with the file name on the FILEDEF for the SYSIN ddname.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue the FILEDEF command or the ASMD command specifying the same file name.

ASMDCMS052E Option list exceeds 512 characters.

Explanation: The string of options that you specified with your ASMD command exceeded 512 characters in length.

System Action: RC=24. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your ASMD command with fewer options specified.

ASMDCMS062E Invalid character *c* **in file name** *file_name*

Explanation: A character that is not permitted was specified in the file name specified on the ASMD command.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Check the format of the option with its correct parameters, and reissue the command with the correct parameter.

ASMDCMS070E Left parenthesis '(' required before option list

Explanation: An option was specified after the file name but before the left parenthesis on the ASMD command.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Issue the ASMD command again with the option specified after the left parenthesis. Only the file name can be specified before the left parenthesis.

ASMDCMS074E Required module *module_name* **MODULE not found**

Explanation: The ASMD command was unable to load the specified module.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Verify you have accessed the disk containing the disassembler and issue the ASMD command again.

ASMDCMS075E Device *device* **invalid for** *file_name*

Explanation: The device specified in your FILEDEF command cannot be used for the input or output operation that is requested in your program. For example, you have tried to read data from the printer or write data to the reader.

Supplemental Information: The variable device name in the text of the message indicates the incorrect device that was specified.

System Action: RC=40. Processing of the command terminates. The system remains in the same status as before the command was entered.

Programmer Response: Reissue your FILEDEF command, specifying the correct device for the required input operation.

Disassembler Messages

ASMD000 Invalid Parameter, specified: parameter

Explanation: The parameter specified is not correct.

Programmer Response: Ensure that the parameter specified is one of the following values:

COPYRIGHTOK
HEX
OPTABLE
NEWNUM
PHASE (VSE Only)
VSESVC

ASMD001 Member and CSECT must be entered via SYSIN

Explanation: The member and CSECT names could not be determined

Programmer Response: Ensure that the module-CSECT statement is present in the control file as the first statement.

ASMD002 Member or CSECT name over 8 characters

Explanation: Either the member name or CSECT name on the module-CSECT statement exceeds 8 characters.

Programmer Response: Correct the member name or the CSECT name on the module-CSECT statement to be 8 characters or fewer.

ASMD003 No member name found on control record

Explanation: The member name is missing from the module-CSECT record.

Programmer Response: Add the desired member name to the module-CSECT record.

ASMD004 Specified member not found

Explanation: The specified member was not found.

Programmer Response: Ensure that the correct member has been specified and:

MVS The correct SYSLIB data set is being used.
VM The correct MACLIB has been referenced with GLOBAL MACLIB.
VSE The correct library has been specified in a LIBDEF search chain.

ASMD005 Specified CSECT not found in member

Explanation: The specified CSECT name was not found in the ESD records for the module.

Programmer Response: Ensure that the correct CSECT has been specified on the module-CSECT statement.

ASMD006 Label table full

Explanation: The disassembler's label table is full.

Programmer Response: Reduce the number of user labels requested in the control file statements.

ASMD007 CLASS name over 16 characters

Explanation: CLASS name in a program object exceeds the maximum length.

System Action: The disassembly is unable to proceed.

Programmer Response: Reduce the length of the CLASS name to 16 characters or less.

ASMD008 The user of the COPYRIGHTOK option ensures that this use is not violate any copyright restrictions or other rights pertaining to the code being disassembled.

System Action: None. Processing continues

Programmer Response: None.

ASMD009 This program may be used to disassemble only object code that you own, or program code for which you have a license to copy and disassemble.

System Action: None. Processing continues.

Programmer Response: None.

ASMD010 A copyright symbol has been found. You should verify that you are allowed to disassemble this object code.

Explanation: The disassembler has detected a copyright symbol within the object code being disassembled.

System Action: The disassembly is stopped

Programmer Response: Ensure that the correct CSECT is being disassembled. Further information on the copyright scans may be found at Chapter 3, “Using the Disassembler” on page 41.

ASMD100 Unidentified record

Explanation: The disassembler is unable to determine the record type.

System Action: The record is ignored and the disassembly stops after all the control statements are read in.

Programmer Response: Provide a valid record type.

ASMD101 Invalid Begin value

Explanation: The data in the Begin field is invalid hex data.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the Begin field.

ASMD102 Invalid End value

Explanation: The data in the End field is invalid hex data, or the value is odd or the value is greater than the length of the CSECT being disassembled.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the End field.

ASMD103 Invalid Register value

Explanation: The data in the register field is invalid hex data, or the value is zero.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide a value between 1 and F in the register field.

ASMD104 Invalid Initial Base value

Explanation: The data in the initial base value is invalid hex data

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the initial base value.

ASMD105 Undefined DSECT

Explanation: The DSECT name specified in columns 25 to 30 is undefined.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the DSECT name to a defined name or provide the missing definition.

ASMD106 Over 256 USING records

Explanation: More then 256 USING records have been specified.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Reduce the number of USING records specified in the control file.

ASMD107 End before Begin

Explanation: The End address specified is less than the Begin address.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change either address so that the End address is greater then the Begin address.

ASMD108 Invalid USING type

Explanation: The USING type specified is not P or D.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the USING type to a valid value.

ASMD109 Invalid User Label name

Explanation: The User Label Name specified has a space as the first character.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the Label Name so that it starts with a non-space character.

ASMD110 Invalid Offset

Explanation: The data in the Offset field is invalid hex data,

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the Offset field.

ASMD111 Invalid Length

Explanation: The data in the Offset field is invalid hex data or the value exceeds 4096.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the Offset field or a valid length value.

ASMD112 Label table overflow

Explanation: The disassembler's label table is full.

Programmer Response: Reduce the number of user labels requested in the control file records.

ASMD113 Invalid DSECT name

Explanation: The DSECT name or field name specified has a space as the first character.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the name so that it starts with a non-space character.

ASMD114 Invalid Number of Fields

Explanation: The number of fields specified is not a valid decimal number.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the number of fields so that it is a valid decimal number

ASMD115 Invalid Offset

Explanation: The data in the Offset field is invalid decimal data or the value exceeds 4096.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid decimal data in the Offset field or a value less than or equal to 4096.

ASMD116 Invalid Length

Explanation: The data in the Offset field is invalid decimal data, or the value exceeds 4096, or the value is zero

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid decimal data in the Offset field or a value between 1 and 4096.

ASMD117 Invalid Begin value

Explanation: The data in the first field (Begin) is invalid hex data.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the Begin field.

ASMD118 Invalid End value

Explanation: The data in the second field (End) is invalid hex data.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Provide valid hex data in the End field.

ASMD119 End Offset before Begin

Explanation: The End offset specified is less than the Begin offset.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change either offset so that the End offset is greater then the Begin offset.

ASMD120 COPY already in progress

Explanation: A COPY member is already being processed.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the use of COPY records so that only one member is being used at a time

ASMD121 Invalid Member name

Explanation: The member name specified in columns 6 to 13 has a space as the first character.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Change the name so that it starts with a non-space character.

ASMD122 COPYLIB OPEN failed

Explanation: The OPEN of the COPYLIB data set failed

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Ensure that the COPYLIB data set has the correct organization and record format.

ASMD123 Member not found on COPYLIB

Explanation: The member was not found in the COPYLIB data set.

System Action: The record is ignored and the disassembly stops after all the control statements are read in and processed.

Programmer Response: Ensure that you specified the correct member and that it does exist as a member within the COPYLIB data set.

ASMD124 Binder API Error RC: *xxxxxx* **Reason:** *xxxxxx***.**

Explanation: An internal error has occurred in the interface between the disassembler and the program object access module.

System Action: The disassembler stops.

Programmer Response: Check the return code in *DFSMS/MVS V1R4 Program Management* SC26-4914 (or later).

ASMD125 Error occurred reading CMS Module.

Explanation: An error occurred when attempting to disassemble a CMS Module.

System Action: The disassembler stops.

Programmer Response: Ensure you have specified the correct CMS Module name when invoking the disassembler.

ASMD126 Module ASMADOP could not be loaded

Explanation: The disassembler could not load common operation code table support module.

System Action: The disassembler stops.

Programmer Response: Contact your systems programmer.

Chapter 4. Using the Program Understanding Tool

Introducing ASMPUT . . . 62
- More about nodes . . . 63
- Getting started . . . 64
  - Other resources . . . 66

Working with ADATA files . . . 67
- Opening an ADATA file . . . 67
- Opening and closing the Control Flow Graph window . . . 67
- Viewing source code . . . 68
  - Changing font properties . . . 68
  - Restoring defaults . . . 69
  - Showing and hiding expanded lines . . . 69
  - Showing and hiding assembly diagnostics . . . 70
  - Showing and hiding analysis messages . . . 71
  - Finding the next assembly diagnostic or analysis message . . . 71
  - Finding text in source code . . . 71
- Viewing ADATA file information . . . 72
- Removing (closing) a file . . . 74

Working with the control flow graph . . . 74
- Expanding and collapsing layers . . . 75
- Adding and removing context . . . 81
- Refreshing and redoing . . . 82
- Hiding and showing return arcs . . . 83
- Marking and unmarking nodes . . . 84
- Opening and closing the Overview window . . . 85
- Zooming . . . 86
- Scrolling . . . 89
- The interaction between source code and the control flow graph . . . 89

ASMPUT windows and window areas . . . 91
- Main window . . . 91
  - Main window file list area . . . 91
  - Main window source code area . . . 92
  - Main window information notebook . . . 92
- Control Flow Graph window . . . 93
- Overview window . . . 95

Restrictions . . . 95

Using online help . . . 96
- Using Topic Help . . . 96
- Using What's This help . . . 97
- The OS/2 help . . . 97

ASMPUT messages . . . 97

© Copyright IBM Corp. 1992, 2004

Introducing ASMPUT

The High Level Assembler Program Understanding Tool (ASMPUT) analyzes assembler language programs, and displays analyzed source code and the corresponding control flow graph.

You can use the control flow graph to trace complex control flows and inter-program linkages.

The control flow graph is made up of nodes and arcs. A node corresponds to a group of lines of code, typically ending with a branch. An arc shows a connection between nodes - a jump, call or return from one line of code to another. (A node that is directly connected to another by an arc is a “linked node.”) For more information, see “More about nodes” on page 63.

You can display different layers of the control flow graph. Higher layers display items in less detail, lower layers reveal items in greater detail. For example, when you expand a node by one layer, ASMPUT breaks the node holding many lines of code into a number of nodes holding fewer lines of code plus connecting arcs.

Apart from isolated nodes, you can trace a path from one node to another, moving along connecting arcs. The nodes immediately joined by arcs to a selected node are the nodes of most importance. Nodes further away are less important to the selected node. When you “remove the context” you remove the more distant nodes. ASMPUT shows only the nodes directly related to the selected node. You can also add context, so that nodes related to those currently displayed will also be displayed.

By adding or removing context, and by expanding or collapsing nodes, you can build a control flow graph that has the degree of simplicity and detail that you want.

Zooming in and out changes the size of the elements in the control flow graph, without making any difference to the structure of the graph. “Working with the control flow graph” on page 74 explains how you can control the display of the control flow graph, such as by zooming and by expanding layers.

Clicking on a line of source code highlights the corresponding node in the graph. Likewise, clicking a node in the graph highlights the corresponding lines of source code. This means that you can trace the control flow either from the source code listing or from the control flow graph, using whichever you find the easiest.

To prepare for using ASMPUT, you must create ADATA files for each program you want to analyze. “Getting started” on page 64 shows how to do this, and outlines the steps for using ASMPUT.

ASMPUT provides information from the High Level Assembler (HLASM) assembly of the programs, for example, listing all the assembly options. “Working with ADATA files” on page 67 lists the procedures for opening and closing ADATA files, and viewing source code and assembly-time information.

ASMPUT has three different windows. The **Main** window shows the ADATA files that are open, information about these files, and a source code listing. The **Control Flow Graph** window shows the control flow graph, and the options and icons you can use the change the structure of the graph. The **Overview** window shows a small copy of the control flow graph, and an area box for quick zoom and

scroll control. “ASMPUT windows and window areas” on page 91 describes these windows.

You can use ASMPUT on Windows or OS/2. The interface and processes of these two versions are essentially the same.

More about nodes

At the lowest level, a node is a sequential group of statements that starts with an entry point, and ends with one or more exit points. None of the statements between the entry point and exit point are in their own right entry or exit points.

The flow of execution through a node means that control enters the node at only one point and leaves the node at one point, via one or more exit paths.

Figure 8 shows a small example. The graphic beside the lines of code is not part of the code, but to help in visualizing the example.

```
stmt#  type of statement

  1.   Executable statement
  2.   Branch to Line 5


  3.   Executable statement
  4.   Executable statement


  5.   Executable statement
  6.   Executable statement
  7.   Conditional branch to Line 1


  8.   Executable statement
  9.   Branch to Line 3
```

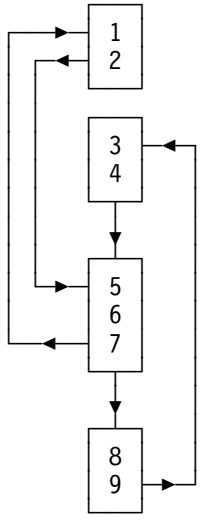


Figure 8. An example of code and nodes

In this example, lines 1 and 2 are a node, with a single entry at line 1 and a single exit via a branch statement at line 2.

Lines 3 and 4 are a node, with entry at line 3 (from line 9) and an exit at line 4 via a fall-through to the next node. The explanatory graphic shows this fall-through explicitly. The node ends at line 4, because line 5 is a branch target, from the branch at line 2. This means that the next node starts at line 5.

Lines 5, 6 and 7 are a node, starting at line 5 and with two exit points at line 7. The two exit points are via a conditional branch and via a fall-through.

Lines 8 and 9 are a node, starting at line 8 and exiting at line 9. The start at line 8 is caused by the fact that the previous node ended at line 7 and the fall-through causes the entry to the new node at line 8.

At the lowest level, a node is always displayed on the control flow graph as a two-dimensional node.

Two-dimensional nodes are aggregated, through layering, to form a single three-dimensional node. For example, a subroutine which is called and returns to the caller may consist of many nodes. The layering process can hide this level of detail, and display the routine as a single three-dimensional node.

Getting started

The steps for using ASMPUT are:

1. **Create an ADATA file**

   You must create an ADATA file on the host before you use ASMPUT, by supplying the ADATA option at assembly time.

   Do not use the XOBJECT or GOFF options, as ASMPUT cannot analyze the resultant output file.

   For more information about these options, see Chapter 3 “Controlling Your Assembly With Options,” of the *Programmer's Guide*.

2. **Download the ADATA file to your PC**

   Download the file to your PC as a binary file, and give it the extension “XAA”.

3. **Start ASMPUT**

   Start ASMPUT by the appropriate means (such as by double-clicking the ASMPUT icon or (in Windows) by selecting from the Start menu). ASMPUT starts with the global values in force when it was last closed, so the position and size of the **Main** window are the same as when ASMPUT was last closed, as are the sizes of the areas within this window, and the showing or hiding of the information notebook, the zoom slider, and return arcs.

4. **Open the ADATA file in ASMPUT**

   The **Open** option of the **File** menu opens a dialog box for you to enter file details. After you complete opening the ADATA file, ASMPUT analyzes it. For more information, see “Opening an ADATA file” on page 67.

   Figure 9 on page 65 shows the **Main** window. This figure, and the following figures in this chapter, show ASMPUT windows while two sample files (CALCPRG.XAA and ADDPRG.XAA) are open. See “Other resources” on page 66 for more information about the sample files. The figures in the PDF and HTML versions of this manual show the ASMPUT windows in color.

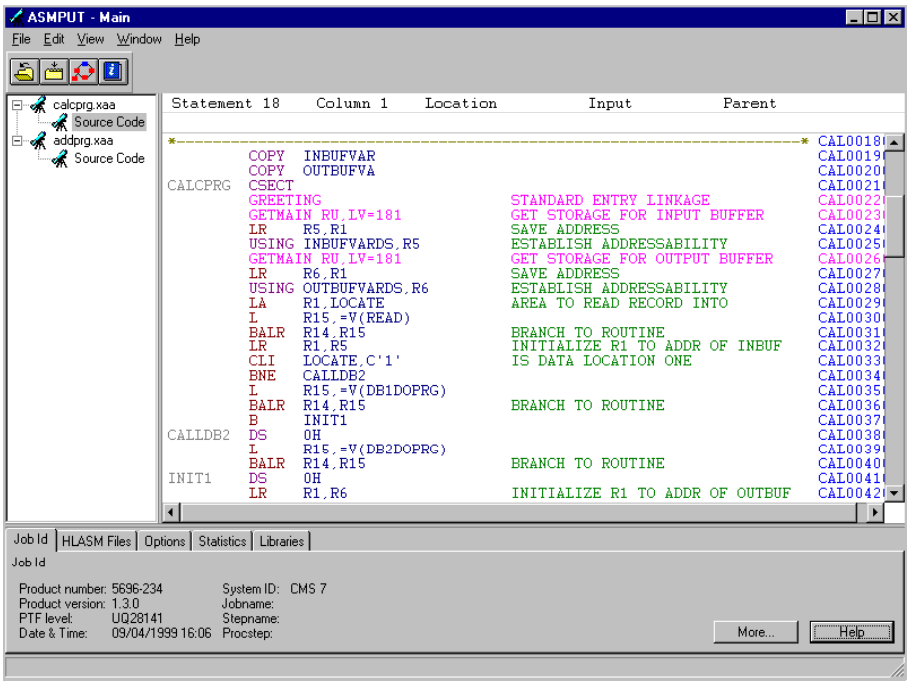


Figure 9. The ASMPUT Main window

5. **Open the Control Flow Graph window**

   ASMPUT displays the control flow graph for all currently opened modules. When you open a new ADATA file, ASMPUT integrates the modules found in that source code into the control flow graph.

   The **Show Graph** option of the **View** menu opens the **Control Flow Graph** window. For more information, see “Opening and closing the Control Flow Graph window” on page 67.

   Figure 10 on page 66 shows the **Control Flow Graph** window.

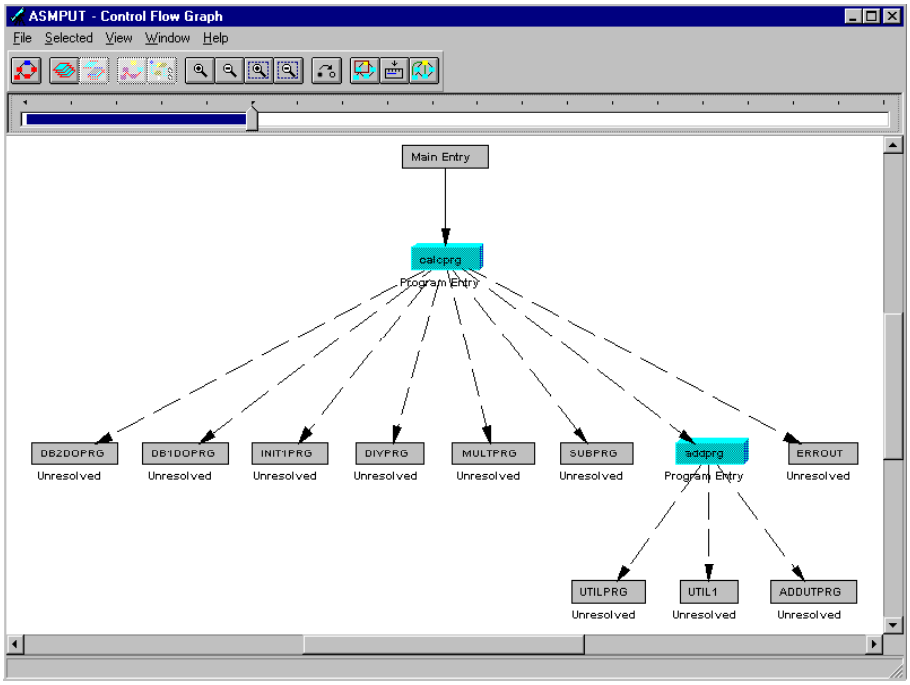


Figure 10. The ASMPUT Control Flow Graph window

6. **Peruse the control flow graph**

   ASMPUT offers many ways to change the appearance of the control flow graph. For example, you can expand the number of layers, or you can remove the context.

   If you click on a node in the graph, the lines of source code corresponding to the node are highlighted in the source code listing.

   For more information, see “Working with the control flow graph” on page 74.

The prime purpose of ASMPUT is to show you a program's control flow in a graphical representation. However, ASMPUT also lets you view source code, and view information created by HLASM when the program was assembled.

Other resources

There are a number of resources to help you use ASMPUT:

- **The slide show demo**

  The slide show demo is a simple demonstration program that runs under Windows (3.1 or Win-OS/2, 95, 98 and NT). It shows screen captures of an ASMPUT session, with annotation. The slide show takes roughly five to ten minutes to run. For a copy of the demonstration and more information, look in the Demo subdirectory.

- **The sample ADATA files**

  The sample ADATA files are in the Samples subdirectory. CALCPRG.XAA is the main program, and most of the other sample programs are secondary programs. For example, open CALCPRG.XAA and look at the control flow graph. There is a gray node for the unresolved external ADDPRG. If you open the file ADDPRG.XAA, the control flow graph is redrawn, and the gray unresolved external node is replaced by the cyan “addprg” program entry node.

- **Online help**

  ASMPUT's online help has a similar structure to this chapter. It provides more detailed step-by-step procedures, and more details for the individual fields of the tabs in the information notebook. The Help structure also has a word search for a word or words in any topic.

  The online help does not include screen captures, and does not provide a detailed listing of ASMPUT messages.

  For more information about online help, see “Using online help” on page 96.

- **The HLASM web site**

  The HLASM web site is at
  http://www.ibm.com/software/awdtools/hlasm/
  As well as the latest HLASM news and downloadable demos, the web site provides the HLASM manuals in HTML and PDF format, ready for online browsing, or downloading to your PC for offline browsing.

Working with ADATA files

ADATA files are produced by HLASM as part of an assembly, when the appropriate options are specified.

The files contain a lot of information about the program. ASMPUT is able to analyze this information, and present it graphically.

You work with existing ADATA files at the Main window (see page 91).

Before ASMPUT can do anything with an ADATA file, you must open it. If you want to remove the contents of the file from the control flow graph, you close it (page 74). While the ADATA file is open, you can view assembly-time information about the file (page 72), look at source code (page 68), and open the control flow graph window (page 67).

Opening an ADATA file

1. On the **Main** window **File** menu, click **Open**.
2. Enter the file name in the **File name** box.
3. Click **Open**.
   ASMPUT analyzes the file, then displays the name of the file in the file list area. If the **Control Flow Graph** window is open, the control flow graph is redrawn (at the highest level of layering), to incorporate any new modules.

You can also start this process by clicking the **Open file** icon on the toolbar.

After you have opened a file, you can view assembly-time information, view source code, and view the control flow graph of the code.

Opening and closing the Control Flow Graph window

The **Control Flow Graph** window shows the control flow graph, and controls to manipulate the graph.

To open the Control Flow Graph window

1. Open at least one ADATA file.

2. Click the **Show Graph** icon on the toolbar, or on the **Main** window **View** menu, click **Show Graph**.
   The **Control Flow Graph** window is opened.

The control options for the **Control Flow Graph** window are described in “Working with the control flow graph” on page 74.

To close the Control Flow Graph window

These methods work from the **Control Flow Graph** window.

- On the **File** menu click **Exit**, or
- Click the **Close** box, or
- Press the F3 shortcut key.
  The **Control Flow Graph window** window is closed.

Viewing source code

1. Open the ADATA file that contains the source code.
   ASMPUT displays the source code for the first source file in the ADATA file.
2. To view the source code of another source file, click another Source Code tag. (Multiple source files are created when there are many assembler source files in the input file to the assembler, and the BATCH option has been specified.)

The source code listing is displayed in the source code area.

After the source code is displayed, you can change the font, show and hide expanded lines from macros and COPY segments, show and hide assembly diagnostics, show and hide analysis messages, find the next assembly diagnostic or analysis message, and look for an item of text in the code.

If the ADATA file containing the source code you want to view is already open, click the Source Code tag of the source code you want to view.

Changing font properties

1. On the **Main** window **Window** menu, click **Fonts**.
2. Select the font, the font style, the font size and any special effects that you want.
3. When all details are acceptable, click **OK**.
   The font changes in the source code area.

The font properties apply only to the display of the source code. Changing the font does not change the font for the file list area, the information notebook, or the nodes of the control flow graph.

The fonts you can change to are the monospaced fonts. They maintain the appearance of the source code listings.

You cannot change the color of the font, since source code items are color coded (for a description of the color code, see “Main window source code area” on page 92).

“Restoring defaults” on page 69 describes how to change back to the original font.

If you make the font smaller, you can see more source lines in the window. If you make the **Main** window larger, the source code area adjusts, to show more source lines.

Restoring defaults

1. On the **Main** window **Window** menu, click **Restore Defaults**.

When you restore defaults, you return the windows to their original size, and return the font in the source code area to its original font.

To change window sizes from their default sizes, drag the window frames.

You can also restore defaults for the **Control Flow Graph** window. Click the **Restore Defaults** option on the **Control Flow Graph** window **Window** menu. This returns the window to its original size.

“Changing font properties” on page 68 describes how to change the font.

Showing and hiding expanded lines

Expanded lines come from COPY segments or macro calls. ASMPUT can either show the expanded lines, or hide them, at your discretion. You can show or hide expanded lines for each particular COPY segment and macro call, or everywhere within the program listing.

Figure 11 shows a listing without any expanded lines shown. (In this figure, the information notebook is hidden, to provide more room for the listing.)

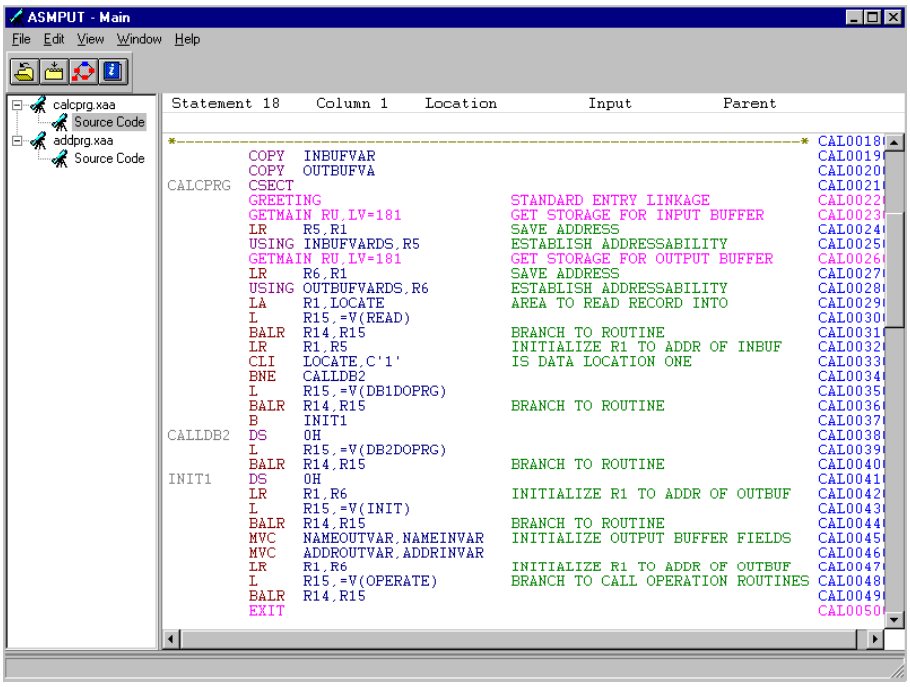


Figure 11. A source code listing not displaying any expanded lines

To show expanded lines

1. Display the relevant source code.
2. Scroll the source code until you find the macro call or COPY segment you want to expand. These are displayed in magenta (pink).

3. Double-click the line with the macro call or COPY instruction in it.

The expanded lines are displayed immediately below the related macro call or COPY instruction, on a gray background.

Figure 12 shows the listing after the fourth line of the listing (the line containing the GREETING macro call) has been double-clicked.

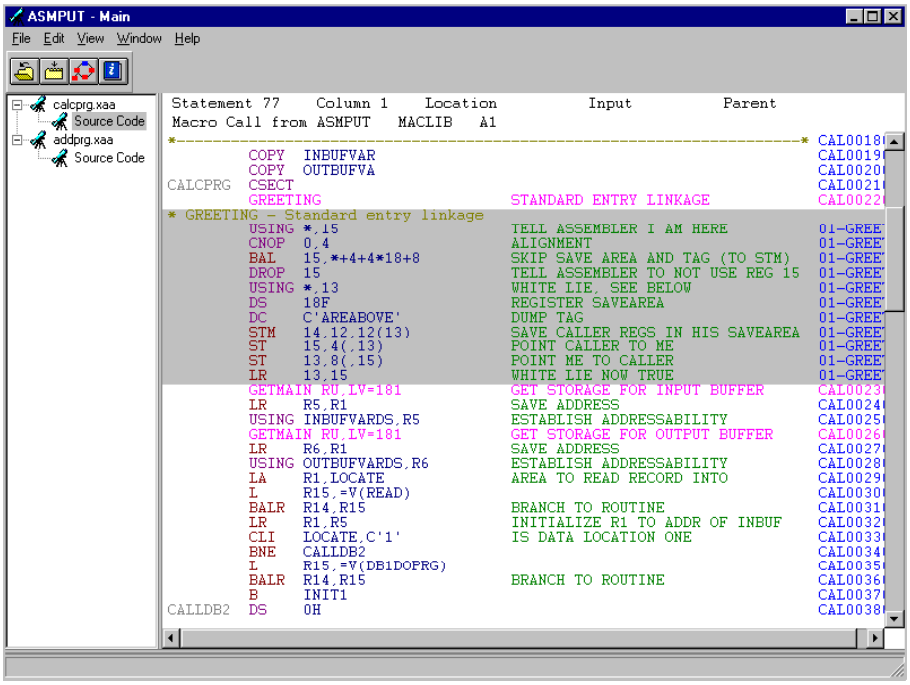

ASMPUT - Main
File Edit View Window Help
calcprg.xaa
Source Code
addprg.xaa
Source Code
Statement 77 Column 1 Location Input Parent
Macro Call from ASMPUT MACLIB A1
COPY INBUFVAR
COPY OUTBUFVA
CALCPRG CSECT
GREETING STANDARD ENTRY LINKAGE
* GREETING - Standard entry linkage
USING *,15 TELL ASSEMBLER I AM HERE
CNOP 0,4 ALIGNMENT
BAL 15,*+4+4*18+8 SKIP SAVE AREA AND TAG (TO STM)
DROP 15 TELL ASSEMBLER TO NOT USE REG 15
USING *,13 WHITE LIE. SEE BELOW
DS 18F REGISTER SAVEAREA
DC C'AREABOVE' DUMP TAG
STM 14,12,12(13) SAVE CALLER REGS IN HIS SAVEAREA
ST 15,4(,13) POINT CALLER TO ME
ST 13,8(,15) POINT ME TO CALLER
LR 13,15 WHITE LIE NOW TRUE
GETMAIN RU,LV=181 GET STORAGE FOR INPUT BUFFER
LR R5,R1 SAVE ADDRESS
USING INBUFVARDS,R5 ESTABLISH ADDRESSABILITY
GETMAIN RU,LV=181 GET STORAGE FOR OUTPUT BUFFER
LR R6,R1 SAVE ADDRESS
USING OUTBUFVARDS,R6 ESTABLISH ADDRESSABILITY
LA R1,LOCATE AREA TO READ RECORD INTO
L R15,=V(READ)
BALR R14,R15 BRANCH TO ROUTINE
LR R1,R5 INITIALIZE R1 TO ADDR OF INBUF
CLI LOCATE,C'1' IS DATA LOCATION ONE
BNE CALLDB2
L R15,=V(DB1DOPRG)
BALR R14,R15 BRANCH TO ROUTINE
B INIT1
CALLDB2 DS 0H

Figure 12. A source code listing displaying a set of expanded lines

If the expanded lines contain a macro, then the inner macro is also expanded.

If expanded lines are highlighted (in the current node), then they are displayed.

To show all expanded lines in the source code area, right-click in the source code area, and from the pop-up menu, click **Show Expanded Lines** so that it is checked.

To hide expanded lines

1. Double-click in the displayed expanded lines, or on the macro call or COPY instruction immediately above the displayed expanded lines.

To hide all expanded lines in the source code area, right-click in the source code area, and from the pop-up menu, click **Show Expanded Lines** so that it is unchecked.

Showing and hiding assembly diagnostics

Assembly diagnostics are messages created by HLASM when the program is assembled.

To show assembly diagnostics

1. Display the relevant source code.

2. Right-click in the source code area.
3. From the pop-up menu, click **Show Assembly Diagnostics**, so that it is checked.

Assembly diagnostics are shown in red on a light gray background, and the message has the prefix “ASMA.”

The **Find Next Diagnostic/Message** option (page 71) takes you to the next assembly error or analysis message.

To hide assembly diagnostics

1. Display the relevant source code.
2. Right-click in the source code area.
3. From the pop-up menu, click **Show Assembly Diagnostics**, so that it is unchecked.

Showing and hiding analysis messages

An analysis message is an indication that ASMPUT has found an instruction that could possibly be in error. It may be worth your while to check the instruction, to make sure that it is correct. The analysis messages are a little like those from a grammar checker in a word processor.

To show analysis messages

1. Display the relevant source code.
2. Right-click in the source code area.
3. From the pop-up menu, click **Show Analysis Messages**, so that it is checked.

Analysis messages are shown as red on a light gray background, and the message has the prefix “ASMP.”

To hide analysis messages

1. Display the relevant source code.
2. Right-click in the source code area.
3. From the pop-up menu, click **Show Analysis Messages**, so that it is unchecked.

Finding the next assembly diagnostic or analysis message

1. Display the relevant source code.
2. Right-click in the source code area. From the pop-up menu, click **Find Next Diagnostic/Message**. Alternatively, press the Ctrl+E keys.

If assembly diagnostics and analysis messages are currently hidden, when you find the next diagnostic or message, it is shown, and stays shown until you hide it.

Finding text in source code

1. Display the relevant source code
2. Right-click in the source code area. From the pop-up menu, click **Find**. Alternatively, press the Ctrl+F keys.
3. Enter the text you want to find in the Find dialog box.
4. If you want an exact match by case, click the **Match case** check box.
5. Click **Find Next**.
   The source code scrolls if necessary, and the matching text is highlighted.

If there are no more occurrences of the text to find, ASMPUT displays the message “ASMP032I End of search.”

To find the next occurrence of text

1. Press the Ctrl+N keys. Alternatively, right-click in the source code area, and from the pop-up menu, click **Find Next**.
   The source code scrolls if necessary, and the matching text is highlighted.
   If there are no more occurrences of the text to find, ASMPUT displays the message “ASMP032I End of search.”

You can find text only if it is displayed in the source code area. If necessary, show expanded lines, assembly diagnostics, and analysis messages before you try and find text.

Finding always starts looking for the text from the current position of the cursor. You can position the cursor by clicking in the source code. You can move the cursor to the start of the source code by pressing the Ctrl+Home keys.

Viewing ADATA file information

The ADATA file information contains information relating to the host assembly of the file.

1. Open the ADATA file.
2. Click the name of the file in the File list area.
3. If necessary, display the information notebook, by clicking the **Show Info Notebook** option of the **Window** menu so that it is checked.
4. Click a tab in the information notebook.

The tabs you can click are:

Job Id Job Information, including details of the Assembler that produced the ADATA file, and when the file was produced.

HLASM Files
Assembler input and output file information. The number of various output files produced, and the list of all the output file names.

Options The Assembler options used at assembly time.

Statistics Statistical information, including the number of records read and written.

Library The number and name of all the libraries read as part of the assembly.

Figure 13 on page 73 shows the **Main** window with the information notebook displayed, showing the Statistics tab.

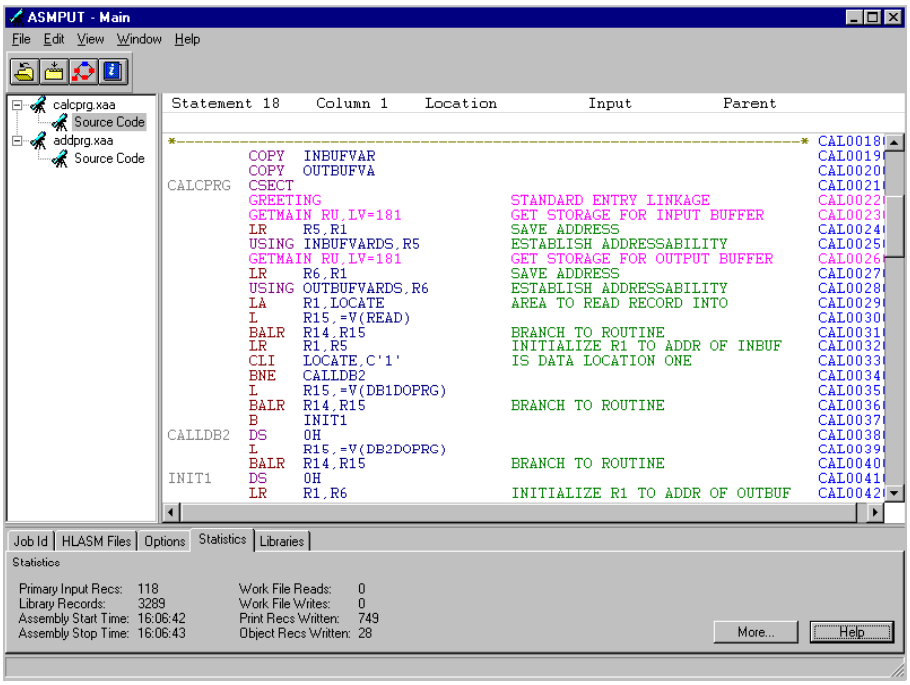


Figure 13. The information notebook Statistics tab

As well as showing names and numbers, each tab has a **More** button. Click this button to see more information, displayed in a new window.

Figure 14 shows the **Statistics Information** More window.

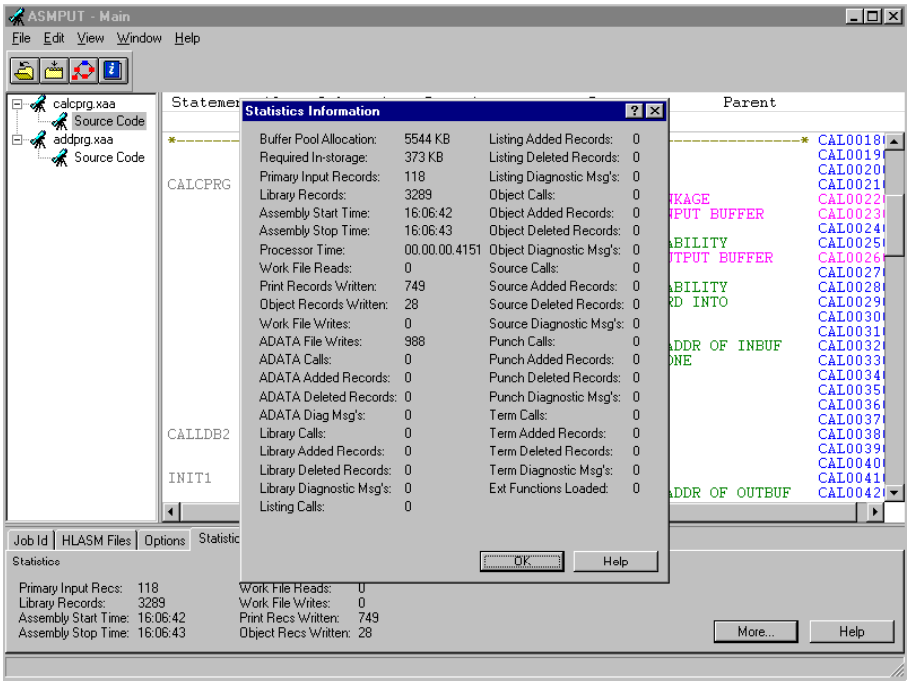


Figure 14. More Statistics Information

Removing (closing) a file

You only need to close an ADATA file if you don't want the contents of the file included in the display control flow graph. Since you are not making changes to any ADATA file, you don't need to close files before you exit from ASMPUT.

1. Right-click the name of the file in the file list area.
2. From the pop-up menu, click **Remove**.

When an ADATA file is closed, its name is removed from the file list area, any displayed source code is no longer displayed, and the control flow graph, if it is displayed, is redrawn.

If you wish to close all ADATA files, but leave ASMPUT open, on the **Main** window **Window** menu, click **Remove All**.

Working with the control flow graph

When you work with the control graph, you can:

- Change the structure
  When you change the structure, ASMPUT adds or removes nodes and arcs.
  - Expand or collapse layers
    When you expand a layer, ASMPUT converts a three-dimensional node into component nodes and arcs. (Three-dimensional and two-dimensional nodes are explained in “Control Flow Graph window” on page 93.)
  - Add or remove context
    When you remove context, ASMPUT no longer displays the nodes that are not directly related to the selected node. These nodes are “removed context.”
  - Refresh or redo
    When you refresh the control flow graph, ASMPUT redraws the top level graph as if you had just opened the **Control Flow Graph** window. When you redo the control flow graph, ASMPUT redraws the current control flow graph, adjusting the zoom level so that the entire control flow graph fits in the control flow graph area.
- Change the appearance
  When you change the appearance, the structure of the control flow graph remains the same, but ASMPUT hides or shows elements, or changes their color.
  - Hide or show return arcs
    The return arcs show returns from calls.
  - Mark or unmark nodes
    A marked node is colored yellow, and remains yellow if expanded or collapsed.
- Change the view
  When you change the view, you make the nodes larger (for easier viewing) or smaller (to see the larger picture), or you look at a different part of the control flow graph.
  - Open and close the Overview window
    The Overview window shows a small copy of the control flow graph. By moving and resizing the area box, you can zoom and scroll.
  - Zoom in and out

Zooming changes the size of the nodes, but not the structure or appearance of the control flow graph.
  - Scroll
    Scrolling changes the part of the control flow graph displayed in the control flow graph area.

- Interact with source code
  When you click on a node, the code corresponding to the node is highlighted in the source code area.

The colors of the nodes and the meaning of name prefixes are explained in "Control Flow Graph window" on page 93.

Expanding and collapsing layers

When the control flow graph is initially displayed, or whenever you open a new ADATA file, the control flow graph shows program entry nodes (cyan nodes) and unresolved external calls (gray nodes), if there are any.

As each layer is expanded, more nodes and more arcs are displayed. The first expansion shows program entry nodes and secondary entry nodes, the second expansion shows all the previous nodes, plus nodes within a program, and so on.

A program entry node holds the primary entry point of the program. This is the default entry point for the program when it is loaded and executed. It is also possible to load a module and start executing it at other entry points. These are secondary entry points, which are held in secondary nodes.

You can expand layers, until there are no more layers to show, and most nodes are green.

You can also reverse the process, to collapse layers. As you collapse a layer, fewer nodes are displayed, until you end up with the initial display, and can collapse no further.

You can expand or collapse one layer or all layers for the entire control flow graph. If you prefer, you can expand or collapse one layer for just one node.

If a node is marked (yellow), when you expand or collapse the node, the resultant nodes (resulting from expanding) or node (resulting from collapsing) retain the marking.

Figure 15 on page 76 shows a completely collapsed control flow graph.

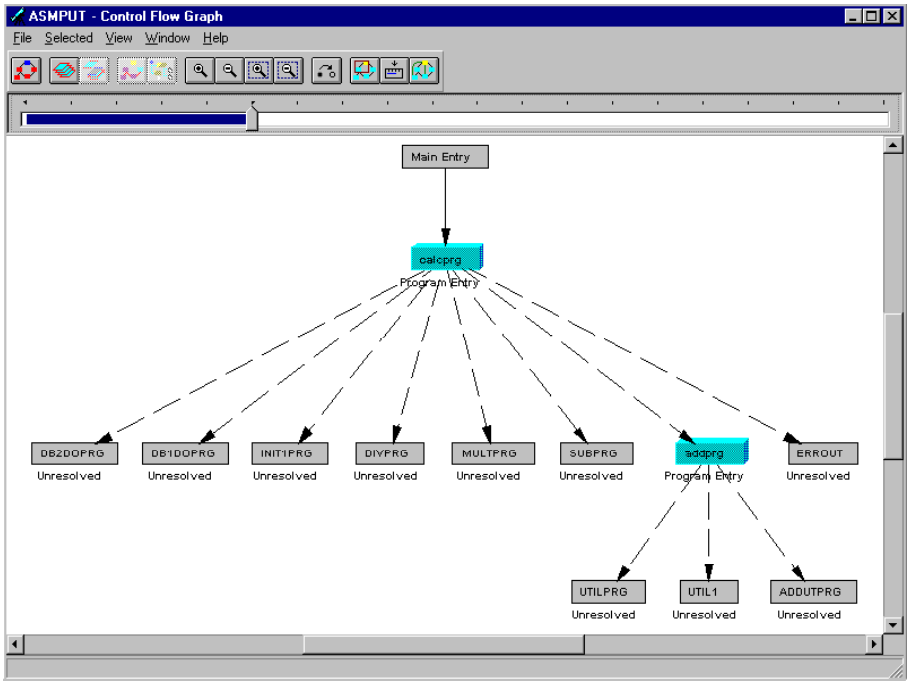


Figure 15. A completely collapsed control flow graph

Expanding one layer for the control flow graph

1. On the **Control Flow Graph** window **View** menu click **Expand Layer**. Alternatively, click **Expand** or right-click on the white area of the graph, and on the pop-up menu click **Expand Layer**.

   When you expand one layer, every three-dimensional node in the control flow graph is expanded by one layer for each node. Individual nodes can be at different layers of expansion. Each node is expanded from its own current layer. Nodes not currently displayed in the control flow graph (because they are in removed context) are not expanded.

In Figure 15 there are eight nodes in the third row of the control flow graph. All but one of these nodes are unresolved external call nodes (they are labeled “Unresolved” beneath each node). The remaining node is a program entry node. In Figure 16 on page 77, which is the same control flow graph after it is expanded by one layer, there are five nodes in the third row of the control flow graph. Three of these nodes are secondary entry nodes (they are the darker, magenta, nodes). They come from the expansion of the program nodes.

Note that in most of the figures, the zoom is adjusted to provide maximum clarity. This means that only a portion of the graph is visible.

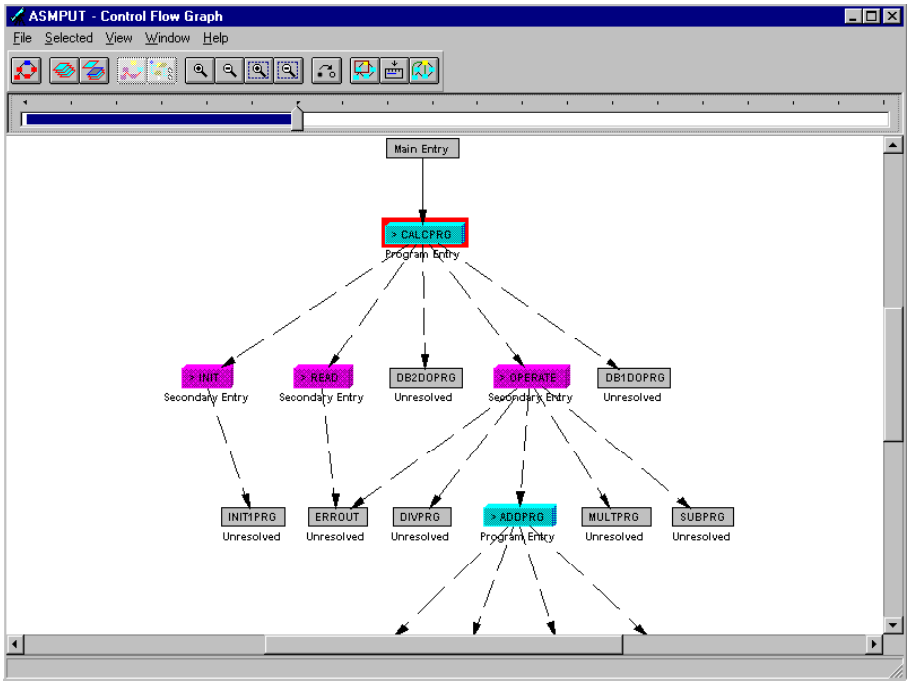


Figure 16. A portion of the same control flow graph expanded by one layer

Expanding all layers for the control flow graph

1. On the **Control Flow Graph** window **View** menu click **Expand All Layers**. Alternatively, right-click on the white area of the graph, and on the pop-up menu click **Expand All Layers**.

   When you expand all layers, every three-dimensional node in the control flow graph is expanded repeatedly, until there are only two-dimensional nodes in the control flow graph. Nodes not currently displayed in the control flow graph (because they are in removed context) are not expanded.

   Some nodes that were connected directly are now connected through intermediate nodes.

Figure 17 on page 78 shows a portion of the same graph, completely expanded. All of the nodes are now two-dimensional.

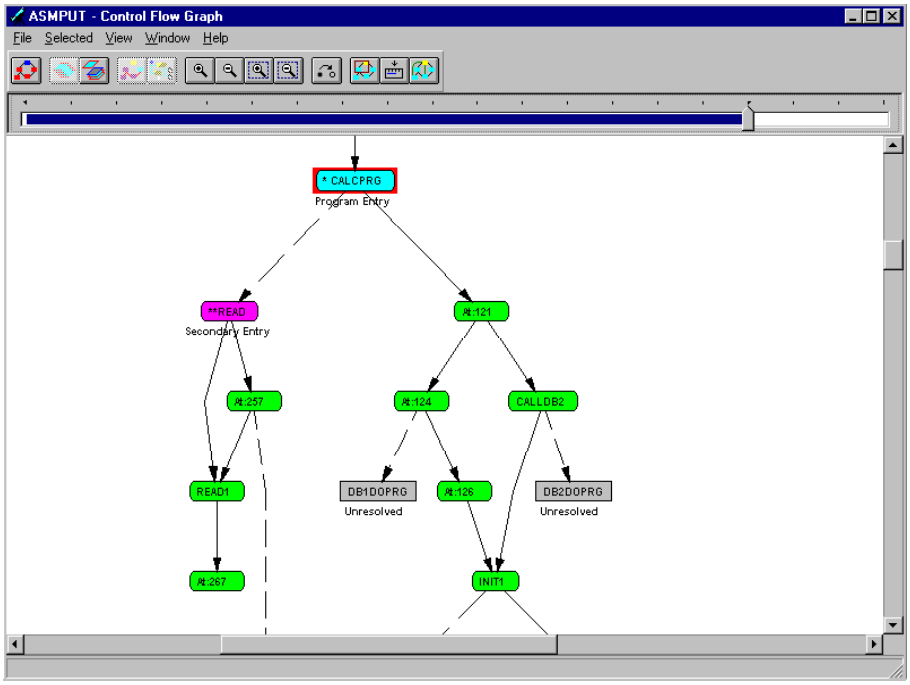


Figure 17. A portion of the same control flow graph, completely expanded

Collapsing one layer for the control flow graph

1. On the **Control Flow Graph** window **View** menu click **Collapse Layer**. Alternatively, click the **Collapse** icon or right-click on the white area of the graph, and from the pop-up menu click **Collapse Layer**.

   When you collapse one layer, every node in the control flow graph is collapsed by one layer for each node, except for nodes that are already completely collapsed. Individual nodes can be at different layers of expansion. Each node is collapsed from its own current layer. If a subordinate node can be collapsed, then a superior node is not collapsed. Nodes not currently displayed in the control flow graph (because they are in removed context) are not collapsed.

Collapsing all layers for the control flow graph

1. On the **Control Flow Graph** window **View** menu click **Collapse All Layers**. Alternatively, right-click on the white area of the graph, and from the pop-up menu click **Collapse All Layers**.

   When you collapse all layers, each node is collapsed to the extent that the control flow graph shows as few nodes as possible, but always shows more than one node.

   For example, if the current control flow graph shows a program entry node and a secondary entry node and nodes for lines of code, then when the control flow graph is collapsed all layers, it shows just the program entry node and the secondary entry node.

   As another example, if the current control flow graph for the same suite of programs shows a few program entry nodes and a few secondary entry nodes and some nodes for lines of code, then when **Collapse All Layers** is clicked, the nodes for the lines of code and the secondary entry nodes are collapsed into the program entry points, because the final control flow graph displays more than one program entry node.

Nodes not currently displayed in the control flow graph (because they are in removed context) are not collapsed.

Expanding one node in context

1. Right-click the (three-dimensional) node you want to expand.
2. From the pop-up menu, click **Expand in Context**.

Alternatively, double-click the (three-dimensional) node you want to expand (double-clicking a two-dimensional node collapses it), or else, if the node is selected, on the **Control Flow Graph** window **Selected** menu click **Expand in Context**.

Figure 18 shows a control flow graph with the ADDPRG node (the second row program entry node) expanded in context. Compare this with Figure 10 on page 66, the same graph before the node was expanded. The difference is that the secondary entry node is now displayed on the third row.

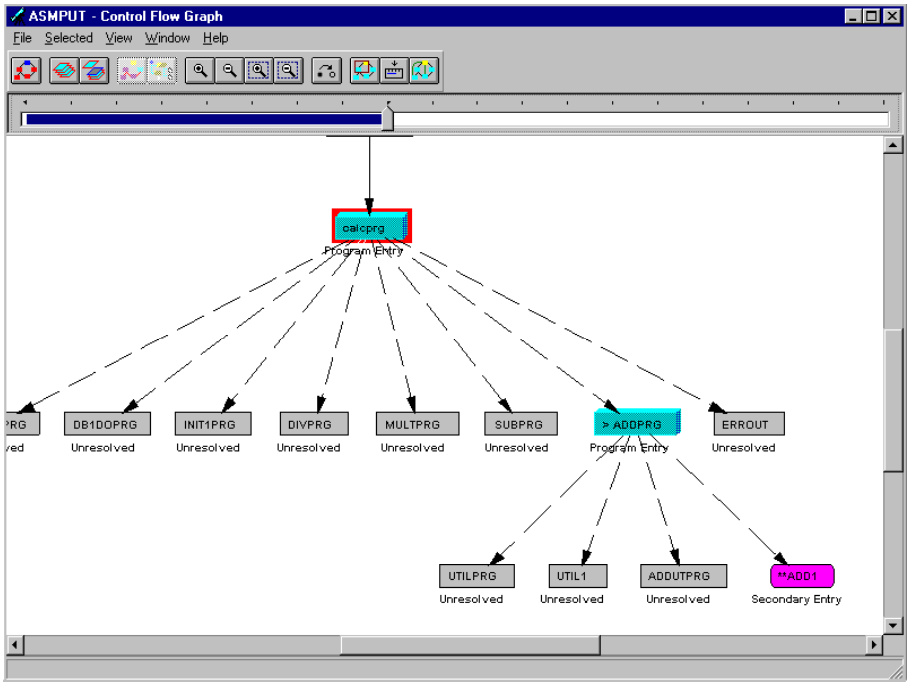


Figure 18. One node expanded in context

When you expand a node in context, the node is expanded one layer. All other nodes remain as they are. The resultant control flow graph is redrawn, to accommodate the additional nodes.

Nodes not displayed in the control flow graph (because they are in removed context), are still not displayed after the node is expanded.

Expanding one node to the window

1. Right-click the (three-dimensional) node you want to expand.
2. From the pop-up menu, click **Expand to Window**.

Alternatively, right-double-click the (three-dimensional) node you want to expand, or else, if the node is selected, on the **Control Flow Graph** window **Selected** menu click **Expand to Window**.

When you expand a node to the window, the node is expanded one layer, and all the context for the node is removed. The resultant control flow graph shows only the nodes and arcs that result from expanding the selected node.

Figure 19 shows the same control flow graph after the ADDPRG node is expanded to the window. The control flow graph displays the nodes that fall within the ADDPRG program. Note that the top node has two lines of text. The second line denotes a call to a secondary entry.

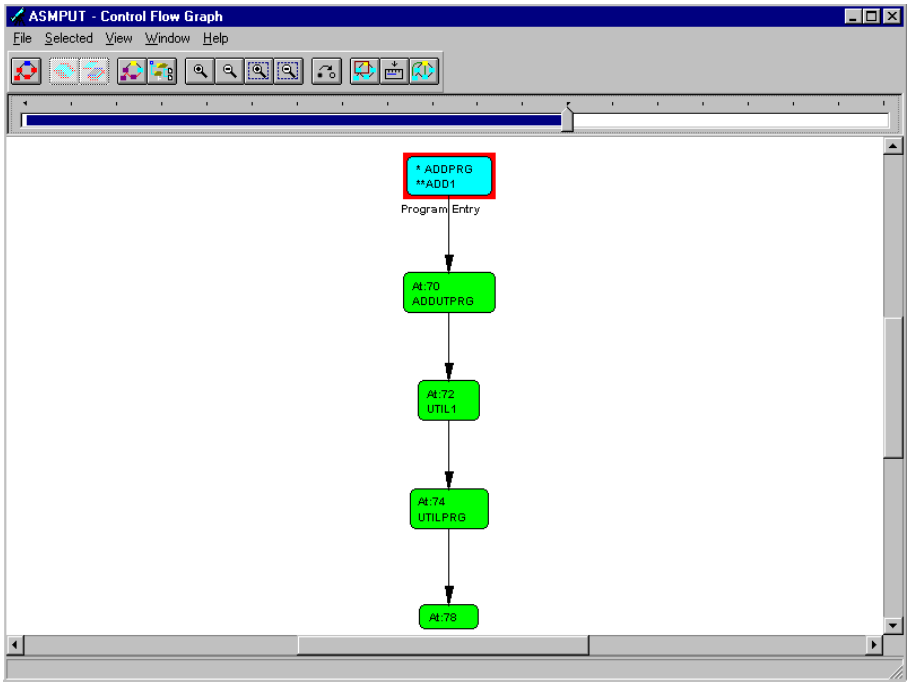


Figure 19. One node expanded to the window

Collapsing in context

1. Right-click the node you want to collapse.
2. From the pop-up menu, click **Collapse in Context**.

Alternatively, double-click the (two-dimensional) node you want to collapse in context (double-clicking a three-dimensional node expands it), or else, if the node is selected, on the **Control Flow Graph** window **Selected** menu click **Collapse in Context**.

When you collapse a node in context, the node is collapsed one layer. All other nodes remain as they are. You cannot collapse a node in context if the resultant control flow graph would only show one node (collapse the node to context instead).

When you collapse in context, nodes that were not displayed in the control flow graph (because they were in removed context) are still not displayed after the node is collapsed.

Figure 20 on page 81 shows the ADDPRG node collapsed to context. The context is just the secondary entry node.

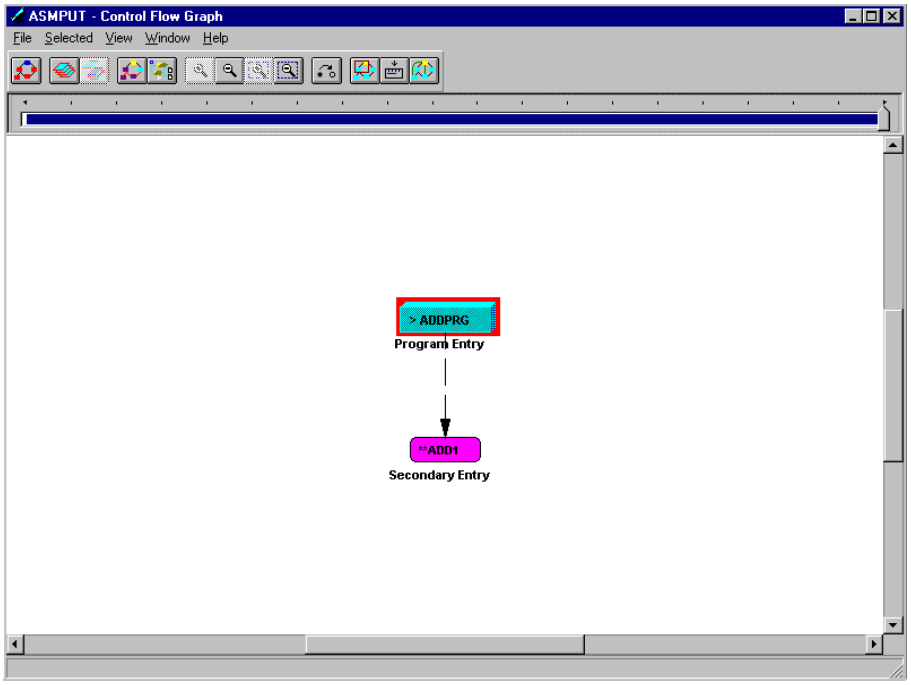


Figure 20. The same node collapsed to context

Adding and removing context

Each node fits within its context. The context changes depends on the current layer that the node is at.

As an analogy, imagine a photograph of two football teams. If you are pointing at one football team, and remove context, the other team is no longer shown. Point at one player within the team, and remove the context, and only that player is shown. Point to that player's mouth, and remove context, and all that is shown is the player's face. You don't have to move through this step by step. Instead, from the full photograph, point at a player's thumb, and remove context, and only the player's hand is shown.

ASMPUT lets you do the same sort of thing with the control flow graph.

When you remove context, ASMPUT shows you just a part of the original control flow graph. This makes it simpler for you to follow the control flow within the displayed elements.

When you add context, you see how the segment of program interacts with other elements in the program, but you add complexity.

Removing context

1. Right-click a node you want to isolate from the context.
2. From the pop-up menu, click **Remove Context**.

Alternatively, if the node is selected, on the **Control Flow Graph** window **Selected** menu click **Remove Context**.

In the resultant control flow graph, some nodes may have two lines of text in them. The first line of text is, as before, the name of the entry point or the line number of the code. The second line of text is the name of a linked node that is not displayed, because it is in the removed context.

You cannot remove context if the resultant control flow graph would only have one node. In this case, consider using the **Expand to Window** option, which expands the selected (three-dimensional) node by one layer, and then removes the context.

Showing context

1. On the **Control Flow Graph** window **View** menu, click **Show Context**.

Alternatively, click the **Show Context** icon or right-click the white area of the graph, and from the pop-up menu click **Show Context**.

This option is not available if there is no removed context.

Redisplayed nodes retain the same layer at which they were removed from the context.

Collapsing to context

1. On the **Control Flow Graph** window **View** menu, click **Collapse to Context**.

Alternatively, click the **Collapse to Context** icon or right-click the white area of the graph, and from the pop-up menu click **Collapse to Context**.

The resultant control flow graph shows all the nodes collapsed into one, and the context for this node. In the previous control flow graph the context was not shown.

Refreshing and redoing

Refreshing the control flow graph displays the graph as if you were opening the **Control Flow Graph** window. The display shows the control flow graph at the top level. All marking is removed, and all nodes are collapsed to the maximum, so that only program entry nodes and unresolved nodes are displayed. The current node (or its collapsed equivalent) remains current.

Redoing the control flow graph displays the graph with the current structure. All the nodes that were displayed in the control flow graph before the graph was “redone” are displayed in the graph afterwards. The graph is displayed zoomed to the minimum magnification, which means that the entire graph is displayed in the control flow graph area. All marking is retained, as is the currently selected node.

In either case, the display or non-display of return arcs is maintained.

Refreshing

1. On the **Control Flow Graph** window **View** menu, click **Refresh**.

Alternatively, click the **Refresh** icon, or right-click the white area of the graph, and from the pop-up menu, click **Refresh**.

Redoing

1. On the **Control Flow Graph** window **Window** menu, click **Layout**, then click **Redo Layout**.

Alternatively, click the **Redo** icon.

Hiding and showing return arcs

Return arcs show how control is returned to a calling program. The return arcs have a dash double dot pattern (—— - - —— - -).

At the highest level, the calling arc and return arc join the same two nodes. However, once layers are expanded, a return arc often connects to a different node to the node that originated the calling arc.

When return arcs are hidden, the control flow graph has fewer lines on it, and so may be easier to follow.

Figure 21 shows a control flow graph with the return arcs hidden.

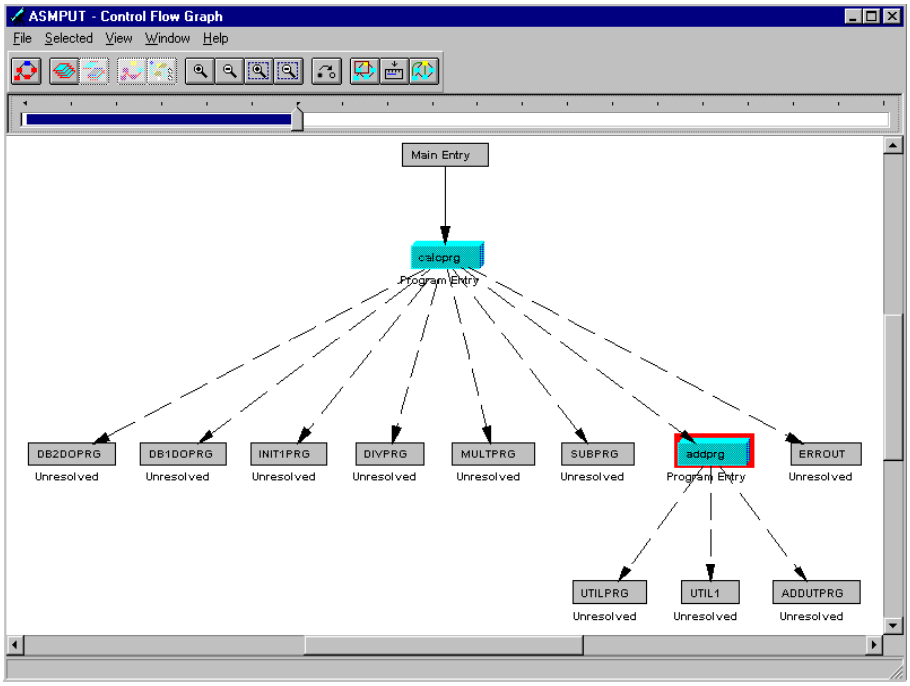


Figure 21. A control flow graph with the return arcs hidden

Hiding return arcs

1. On the **Control Flow Graph** window **Window** menu, click **Show Return Arcs** so that is it unchecked. Alternatively, click the **Show Return Arcs** icon. The control flow graph is redrawn, hiding the return arcs. Any marked nodes remain marked, and the currently selected node stays selected.

Showing return arcs

1. On the **Control Flow Graph** window **Window** menu, click **Show Return Arcs** so that is it checked. Alternatively, click the **Show Return Arcs** icon. The

control flow graph is redrawn, showing the return arcs. Any marked nodes remain marked, and the currently selected node stays selected.

Figure 22 shows the same graph with the return arcs displayed.

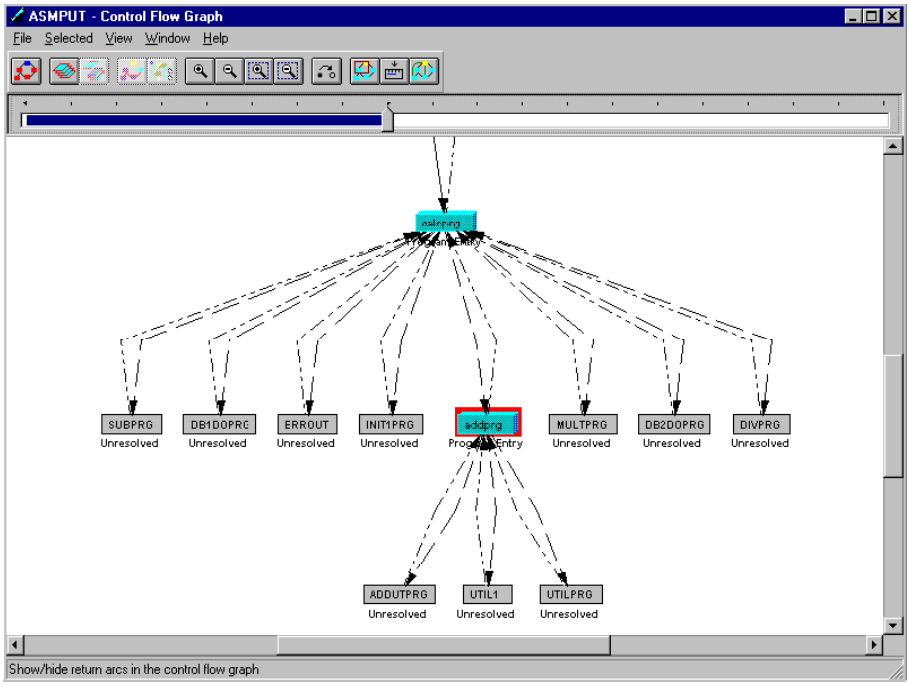


Figure 22. A control flow graph with the return arcs displayed

Marking and unmarking nodes

When a node is marked, it is colored yellow. There is no other change to the control flow graph. When a node is unmarked, the color of the node changes to reflect the status of the node (program entry, secondary entry, and so on).

If a node is marked, and is subsequently expanded, then all the expanded nodes are also marked. If a node is marked, and is subsequently collapsed, then the collapsed node is also marked.

Marking a node

1. Select the node you want to mark.
2. On the **Control Flow Graph** window **Selected** menu click **Mark**.

Alternatively, right-click a node you want to mark, and from the pop-up menu click **Mark**.

Unmarking a marked node

1. Select the node you want to unmark.
2. On the **Control Flow Graph** window **Selected** menu click **Unmark**.

Alternatively, right-click a node you want to mark, and from the pop-up menu click **Unmark**.

Unmarking all marked nodes

1. On the **Control Flow Graph** window **Selected** menu click **Unmark All**.

Opening and closing the Overview window

The **Overview** window shows all of the control flow graph. The shaded area box shows the part of the control flow graph currently displayed in the control flow graph area. By moving and resizing the area box, you can change the contents of the control flow graph area.

To open the Overview window

1. Open the control flow graph.
2. Click the **Show Overview** icon on the toolbar, or on the **Control Flow Graph** window **Window** menu click **Show Overview** so that the option is checked. The **Overview** window is opened.

The ways in which the Overview window can be used to control zooming and scrolling are described in “Zooming” on page 86 and “Scrolling” on page 89.

Figure 23 shows the **Overview** window displayed.

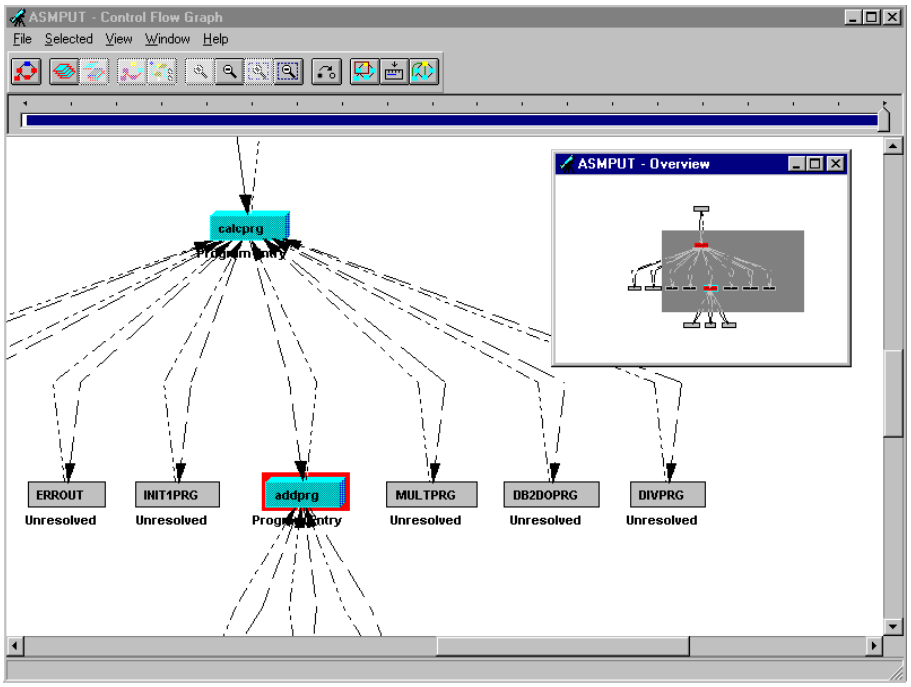


Figure 23. The Overview window

To close the Overview window

These methods work provided the **Overview** window is currently open.

1. Click the **Show Overview** icon on the toolbar, or on the **Control Flow Graph** window **Window** menu click **Show Overview** so that the option is unchecked, or click the **Close** box of the **Overview** window.

Zooming

Zooming makes no change to the structure or color of the control flow graph. Instead, it changes the size of nodes and lettering on the control flow graph.

When you zoom in towards maximum zoom, items become larger, which means that you see less of the complete control flow graph, but text is easier to read.

Figure 24 shows part of a control flow graph at maximum zoom.

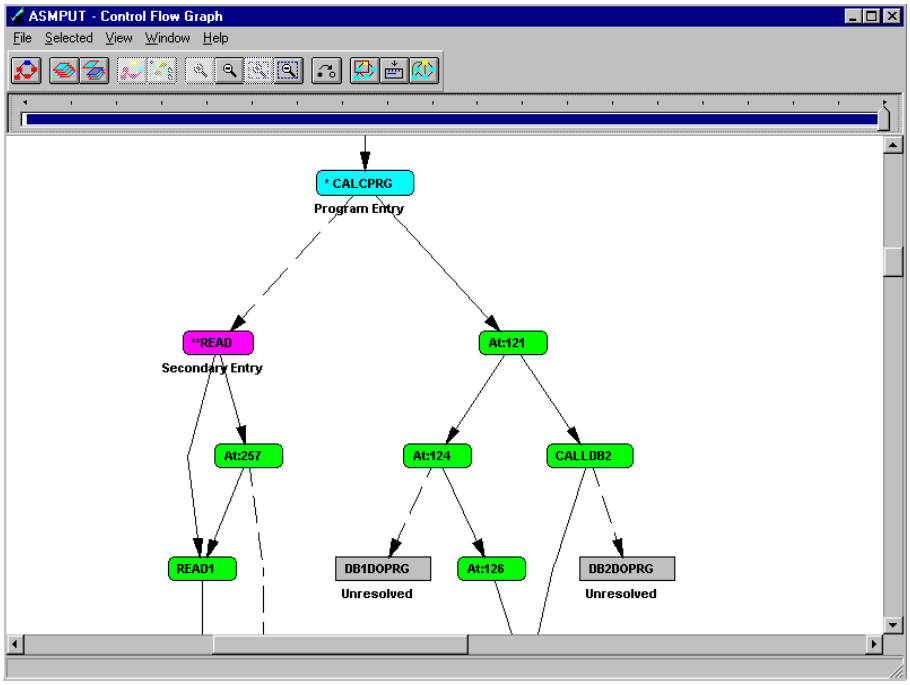


Figure 24. A graph at maximum zoom

When you zoom out towards minimum zoom, items become smaller, so you see more (or all) of the complete control flow graph, but text becomes harder to read.

Figure 25 on page 87 shows part of the same graph at minimum zoom. The node that is second from the top corresponds to the node at the top of the previous figure.

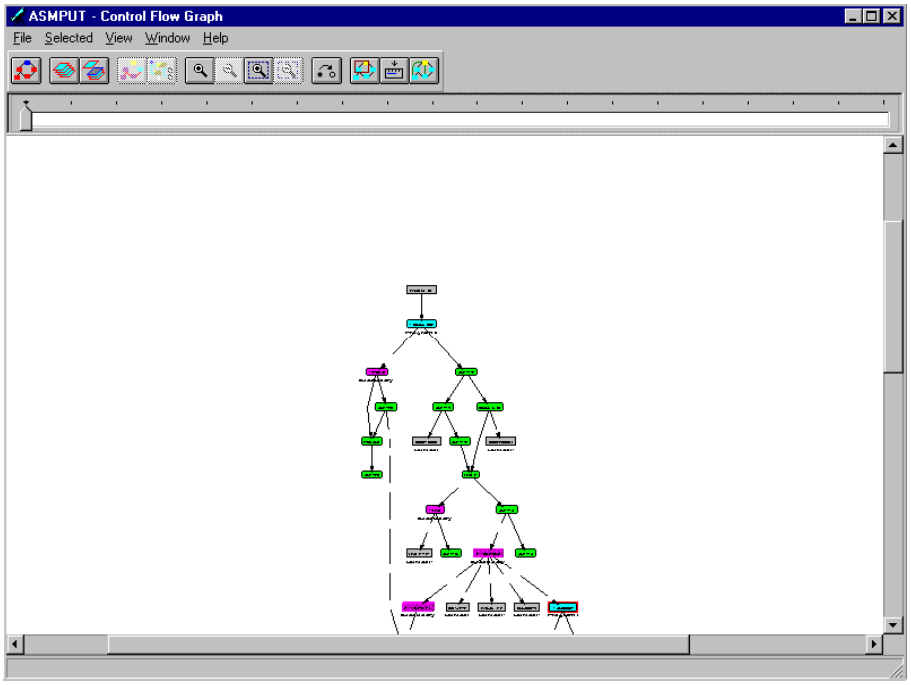


Figure 25. A graph at minimum zoom

The graph displayed in the **Overview** window remains the same size when you zoom. However, the area box changes size. As you zoom in it becomes smaller, and as you zoom out it becomes larger.

At minimum zoom, the entire control flow graph fits in the control flow graph area.

If the control flow graph has only a few elements, you may find that the largest size and the smallest size are the same. The intermediate levels of zoom provide magnifications between those offered by the maximum and minimum zoom.

The zoom slider, located between the toolbar and the control flow graph area, shows the current level of zoom. You can also use it to adjust the level of zoom. Before you can use the zoom slider, make sure it is shown. If you hide the zoom slider, you increase the size of the control flow graph area.

Showing the zoom slider

1. On the **Control Flow Graph** window **Window** menu, click **Show Zoom Slider** so that it is checked. Alternatively, click the **Show Zoom Slider** icon on the toolbar.

Hiding the zoom slider

1. On the **Control Flow Graph** window **Window** menu, click **Show Zoom Slider** so that it is unchecked. Alternatively, click the **Show Zoom Slider** icon on the toolbar.

Zooming in

- On the **Control Flow Graph** window **View** menu, click **Zoom In**, or
- On the **Control Flow Graph** window **View** menu click **Zoom In Rectangle**, then click in the control flow graph, then drag to form an outline rectangle, or

- Click the **Zoom In** icon, or
- Drag the zoom slider to the right, or
- On the **Control Flow Graph** window **Selected** menu click **Zoom In On**, or
- Right-click a node, and from the pop-up menu click **Zoom In On**, or
- Click on the edge of the area box in the **Overview** window and shrink the area box, or
- Click the **Zoom In Rectangle** icon, then click in the control flow graph, then drag to form an outline rectangle.

When you click **Zoom In On**, maximum zoom is applied. When you click **Zoom In**, the zoom goes up by one level. When you zoom in by shrinking the size of the area box on the **Overview** window or by clicking **Zoom In Rectangle**, you can zoom to intermediate levels.

When you click **Zoom In Rectangle** and then drag a rectangle, the area you select is magnified to fill the control flow graph area. When you drag to select the area, the rectangle keeps the same proportions as the control flow graph area.

If you are already at maximum zoom, the options and icon are not available.

When you click **Zoom In On** or **Zoom In Rectangle**, you zoom and scroll at the same time.

Zooming out

- On the **Control Flow Graph** window **View** menu click **Zoom Out**, or
- Click the **Zoom Out** icon, or
- Drag the zoom slider to the left, or
- On the **Control Flow Graph** window **Selected** menu click **Zoom Out From**, or
- On the **Control Flow Graph** window **View** menu click **Zoom Out Rectangle**, then click in the control flow graph, then drag to form an outline rectangle, or
- Right-click a node, and from the pop-up menu, click **Zoom Out From**, or
- Click on the edge of the area box in the **Overview** window, and expand the area box, or
- Click the **Zoom Out Rectangle** icon, then click in the control flow graph, then drag to form an outline rectangle.

When you click **Zoom Out From**, minimum zoom is applied. When you click **Zoom Out**, the zoom goes down one level. When you zoom out by expanding the size of the area box on the **Overview** window, or click **Zoom Out Rectangle**, you can zoom to intermediate levels.

If you are already at minimum zoom, the options and icon are not available.

Scrolling

Scrolling shows a different part of the control flow graph in the control flow graph window area.

You can scroll mechanically, using the scroll bars or the area box, or you can scroll by selecting an option.

Scrolling mechanically

- On the **Control Flow Graph** window, click and drag the horizontal scroll bar or the vertical scroll bar, or
- On the **Control Flow Graph** window, right-click the point you want to move, drag the mouse pointer to the new position, and release the mouse button, or
- On the **Overview** window, click on the area box and drag it to a new position.

Scrolling by option

- Right-click an arc, and from the pop-up menu click **Scroll to Source** to select the node that is at the tail of the arc, or click **Scroll to Target** to select the node that is at the head of the arc. The control flow graph scrolls so that the selected node is in the window area.

  Alternatively, double-clicking the arc is the same as clicking **Scroll to Target**, and right-double-clicking the arc is the same as clicking **Scroll to Source**.

  Alternatively, if an arc is selected, on the **Control Flow Graph** window **Selected** menu click **Scroll to Target** or **Scroll to Source**.
- Right-click a node, and from the pop-up menu, click **Center On**. The control flow graph scrolls so that the selected node is in the center of the window area. Alternatively, if a node is selected, on the **Control Flow Graph** window **Selected** menu click **Center On**.

When you zoom in on a node it is centered.

When you click on lines of source code, the control flow graph is scrolled until the selected node is displayed in the control flow graph window area. See “The interaction between source code and the control flow graph” for more information.

The interaction between source code and the control flow graph

When you click on a node in the control flow graph (and thus select the node), ASMPUT highlights the corresponding lines of code in the source code listing. When you click on a line of code in the source code listing, ASMPUT selects the corresponding node in the control flow graph.

This means you can work between the control flow graph and the source code listing, to better understand your program.

To gain the most benefit from this, size and position the **Main** and **Control Flow Graph** windows so that both appear on your screen at the same time (preferably with no overlap).

Figure 26 on page 90 shows the **Control Flow Graph** window at the left of the screen, and the **Main** window at the right of the screen. The second node in the

control flow graph is the selected node. The highlighted code in the source area is the code that corresponds to this node.

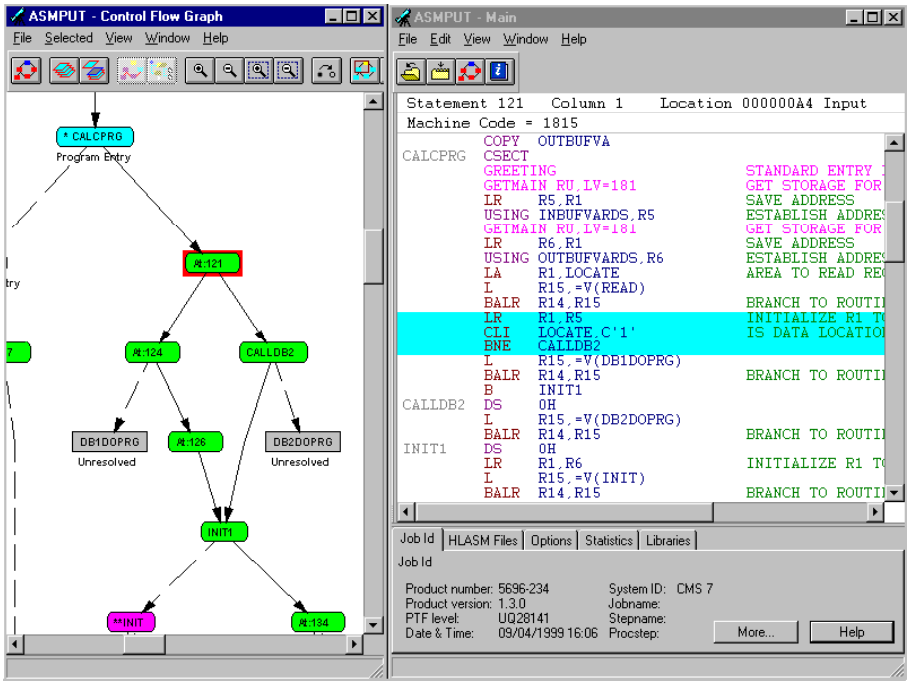


Figure 26. Displaying the Control Flow Graph window and Main window side-by-side

If you select a node on the control flow graph, then the relevant source code is always highlighted. However, if you click on a line in the source code, the corresponding node may not be in the current control flow graph. In this case, the highlight may move unexpectedly, not settling on the lines you clicked. To overcome this, if you want to select a node by clicking on a line of source code, make sure that you are showing a control flow graph of all programs, expanded to all layers.

If you click on a selected node, and thus unselect it, then the highlighting disappears from the source code listing.

Highlighting source code from the control flow graph

1. Right click a node.
   The related code is highlighted in cyan in the source code listing, and the listing is scrolled so that the highlighted text is in the middle of the source code area.

If you click a three-dimensional node, the highlighting is only for the lines that correspond to the top two-dimensional node currently collapsed into the three-dimensional node.

If you click a node, and the code for the node was in a set of hidden expanded lines, then the lines are shown while the node is selected, but hidden after the node is deselected.

Selecting a node from the source code

1. Click a line of source code.

The related node is selected, and the code that corresponds to the top node is highlighted in cyan.

If the node is three-dimensional, the highlighted lines do not necessarily include the line you clicked. Instead, they are the lines that correspond to the top two-dimensional node if the three-dimensional node was expanded to its fullest. There is a guaranteed correspondence only if the node is two-dimensional.

ASMPUT windows and window areas

ASMPUT has three different windows.

The **Main** window shows the ADATA files that are open, information about these files, and a source code listing.

From the **Main** window, you can open a **Control Flow Graph** window. This window shows part or all of the control flow graph, which is for all open ADATA files. In the **Control Flow Graph** window you look at the control flow graph and change its structure.

The **Overview** window is a subsidiary of the **Control Flow Graph** window. It shows all of the control flow graph in a compressed form. The control box provides quick zoom and scroll control.

Main window

The **Main** window displays text information about open XAA files. The window has three areas:

File list area
The list of open ADATA files

Source code area
Where the source code of one source code file is listed

Information notebook
Tabs with more information about an ADATA file

From the **Main** window, you control the opening and closing of ADATA files, the viewing of source code, and the opening of the **Control Flow Graph** window. For details, see “Working with ADATA files” on page 67.

You can adjust the relative sizes of the areas by dragging the thin lines between them. When you change the size of the **Main** window, the areas change size proportionately.

Main window file list area

The file list area is at the left side of the **Main** window. The file list area displays a list of the open ADATA files. Files are listed in the order in which they were opened.

A particular file is highlighted by clicking on its name. Information about the highlighted file is displayed in the information notebook.

The source code file tags in a program are listed by clicking the + sign to the left of the file name. The source code for a source code file is listed in the source code area when the Source Code tag is clicked.

Main window source code area

The source code area is at the right side of the **Main** window. The source code area displays the source code that is part of an ADATA file.

The source code is color coded:

Dark red
: Machine instructions

Violet
: Assembler instructions

Navy blue
: Machine and assembler instruction operands

Dark green
: Remarks

Brown
: Comments

Magenta (pink)
: Macro calls and COPY segments

Gray
: Labels

Blue
: Sequence numbers

Red on a light gray background
: Assembly diagnostics (HLASM) or analysis messages (ASMPUT)

A light gray background indicates a diagnostic, a message, a macro expansion, or a COPY segment expansion.

A cyan (light blue) background is the highlight applied to the lines of code in the current node. The highlight is only applied to executable lines of code, so non-executable lines of code may be interspersed between the highlighted lines of code. The highlight is applied even if the **Control Flow Graph** window is not open. However it is not applied if the **Control Flow Graph** window is open, but no node is currently selected.

In the source code area you can display and hide expanded lines, find text, and change the font.

Main window information notebook

The information notebook is at the bottom of the **Main** window. (If it is not shown, display it by clicking **Show Info Notebook** on the **Window** menu, so that it is checked, or by clicking the **Show Notebook** icon on the toolbar.) The information notebook displays assembly-time information about an ADATA file.

By selecting a tab, you can look at Job Id information, HLASM file information, assembly options, assembly statistics, and library call information.

The online help has a topic for each tab. The help information includes a description of each field in the tab. In Windows, you can also get “What's This” help on the **More** panels by clicking the question mark (“?”) and then clicking a field on the **More** panel.

Control Flow Graph window

The **Control Flow Graph** window displays the control flow graph. The control flow graph is displayed in the control flow graph area, which is the white area in the center of the window. A vertical and a horizontal scroll bar provide a means of moving the control flow graph around in the control flow graph area. The zoom slider (if shown) above the control flow graph area provides a means of zooming the control flow graph. A menu and a toolbar are at the top of the window.

The control flow graph is a set of nodes and arcs.

A node is displayed as a rectangle. It corresponds to a contiguous group of lines of source code (see “More about nodes” on page 63 for more information).

An arc connects two nodes. It leaves from a source node, and points to a target node.

Various controls at the edge of the window let you adjust the appearance of the control flow graph.

Nodes

Nodes can have different appearances and different colors.

The different appearances are two-dimensional nodes and three-dimensional nodes. Three-dimensional nodes have three faces. The front face is rectangular, and the three faces have different shadings, so that the node looks like a three-dimensional rectangular box. The two-dimensional nodes have only one face, a rectangle with rounded corners.

These nodes have this meaning:

Two-dimensional
: The node cannot be expanded any more.
If you collapse this node, it is replaced by a three-dimensional node, and some nodes and arcs disappear.

Three-dimensional
: The node can be expanded.
When it is expanded, the node is replaced by a group of nodes and arcs. Some of these nodes may in turn be three-dimensional, and so can be expanded. However, eventually, every node is two-dimensional, which means that the control flow graph cannot be expanded any more.

The different colors have these meanings:

Gray
: An unresolved external call.

Cyan (light blue)
: A program entry point

Magenta (pink)
: A secondary entry point

Yellow
: A marked node

Green
: Any other node

Sometimes the name in a node has a prefix. The prefixes have these meanings:

> Three-dimensional cyan or magenta node containing an entry

< Three-dimensional cyan node containing a program

***** Two-dimensional cyan program entry

****** Two-dimensional magenta secondary entry

When the context is removed, nodes may have two lines of information. The first line is the name of the node. The second names the call to a node in the surrounding removed context. The prefix to the name provides further information.

The currently selected node is surrounded by a red highlight. The color of the node does not change.

If you mark a three-dimensional node, and then expand it, the nodes it expands into remain yellow, indicating that they are marked, and any arcs that link two marked nodes become pink.

If you mark a node, and then collapse that node in context, the resultant three-dimensional node remains yellow, indicating that it is marked. (However, if you now expand that node, only the node you have previously marked is shown as marked.)

If you open a new ADATA file which is able to resolve an external call, then, in the redrawn control flow graph, the node changes color, since the call becomes resolved.

In general, a node is connected to another node or nodes. However, if the node represents stand-alone code, then the node may be neither the target nor the source for an arc.

Arcs

Arcs connect nodes. An arc must go from one node (the source node), to another node (the target node).

Arcs have this appearance:

A long dash (—— ——)
An internal or external call

A long dash followed by two short dashes (—— - - —— - -)
A return from a called routine

A solid line (————)
Everything else

Any arc joining two marked nodes is magenta. An arc selected by clicking it is red. Any other arc is black.

Controls

The zoom slider above the control flow graph area controls the level of zoom. Moving it to the left makes graph elements smaller (zoom out). When graph elements are at their smallest, the entire control flow graph fits into the control flow graph area. Moving it to the right makes graph elements larger (zoom in).

If the control flow graph does not fit completely into the graph area, you can use the horizontal and vertical scroll bars to move around the graph.

For information about what you can do on the **Control Flow Graph** window, see "Working with the control flow graph" on page 74.

Overview window

The **Overview** window displays the entire control flow graph, at a much-diminished size.

The structure of the control flow graph, the color of nodes, and the appearance of arcs, are accurately reflected in the overview graph, and the selected node is displayed with a red outline. However, the overview graph displays no lettering.

The gray rectangle on the overview graph is the area box. It indicates which part of the control flow graph is currently displayed in the control flow window. All colors under the area box change to complementary colors. For example, cyan changes to red, and green changes to magenta.

The area box has the same proportions as the display area of the control flow graph. If you resize the control flow graph window, the area box changes size.

You can move the area box around, by clicking and dragging it. When you do so, you effectively scroll the contents of the control flow graph window.

You can change the size of the area box by clicking and dragging the edge of the box. When you do so, you zoom the contents of the control flow graph window.

If you use other means to zoom or scroll, the area box is moved or resized in response.

When you resize the **Overview** window, the overview graph is resized. By this means you can enlarge or shrink the overview graph.

For information about opening and closing the **Overview** window, see "Opening and closing the Overview window" on page 85.

Restrictions

When you use ASMPUT, the following restrictions apply:

- The analysis engine cannot deal with branches involving an index register. This means that code using branch tables is not analyzed correctly.
- The analysis of register usage requires further improvement. The results displayed by this version may not be correct in all instances.
- It is not possible to analyze any program that specifies a non-zero entry point on an END statement which involves an expression for the END statement operand. The use of a label symbol by itself is supported.
- The Graphic Print function:
  - May have problems in font selection varying in different printers.
  - In multi-page mode, divides a graphic into six pages regardless of the size of the graphic.

- Only prints to A4 paper size.

- The Graphic Export function exports a graphic to a fixed-size BMP file.
- The Graphic Print function is not supported for Windows 2000 or later.
- Push buttons do not function on Windows 2000 or later.
- ASMPUT only supports HLASM Release 4 format ADATA records. If you use HLASM Release 5 to produce the ADATA file, then the assembly must use the sample ADATA exit ASMAXADR to reformat to Release 4 format (ASMAXADR is a sample source installed as part of HLASM for MVS and VM users).

Using online help

The Windows version of ASMPUT has two forms of online help. The first is topic help, which you access by clicking **Help Topics** or **Index** on the **Help** menu, or by clicking a **Help** button. When you do this, ASMPUT opens the help file, which gives you access to many topics. The second is “What's This” help. This is accessible from the **More** windows of the information notebook tabs, and a few dialog boxes, and shows you a note about the item you point at.

Using Topic Help

When you invoke topic help, ASMPUT displays the topic help file (Figure 27).

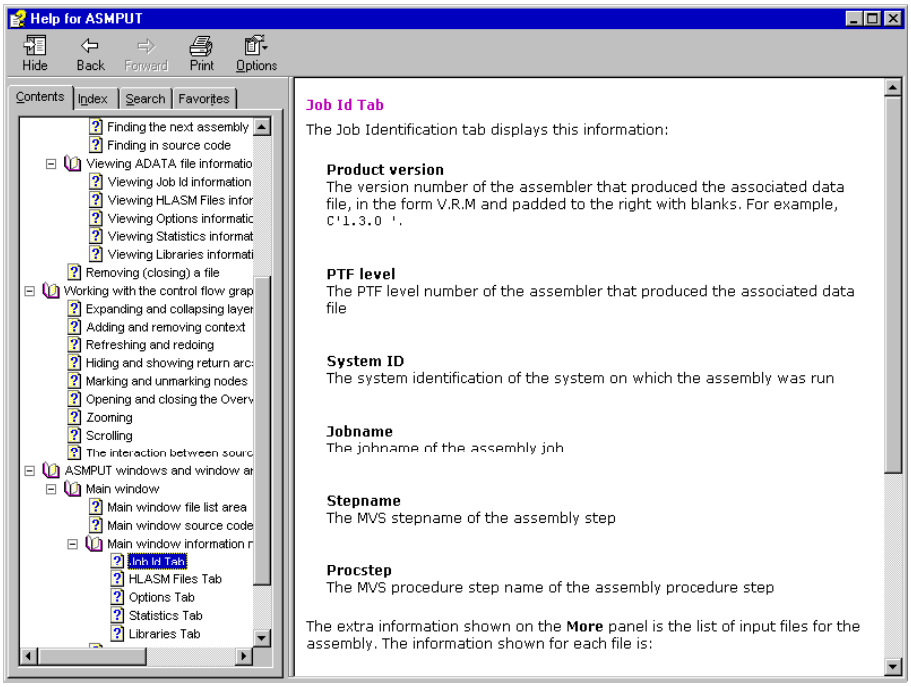

Help for ASMPUT
Hide
Back
Forward
Print
Options
Contents
Index
Search
Favorites
Finding the next assembly
Finding in source code
Viewing ADATA file informatio
Viewing Job Id information
Viewing HLASM Files infor
Viewing Options informatic
Viewing Statistics informat
Viewing Libraries informati
Removing (closing) a file
Working with the control flow grap
Expanding and collapsing layer
Adding and removing context
Refreshing and redoing
Hiding and showing return arc
Marking and unmarking nodes
Opening and closing the Overv
Zooming
Scrolling
ASMPUT windows and window ar
Main window
Main window file list area
Main window source code
Main window information
Job Id Tab
HLASM Files Tab
Options Tab
Statistics Tab
Libraries Tab
Job Id Tab
The Job Identification tab displays this information:
Product version
The version number of the assembler that produced the associated data file, in the form V.R.M and padded to the right with blanks. For example, C'1.3.0 '.
PTF level
The PTF level number of the assembler that produced the associated data file
System ID
The system identification of the system on which the assembly was run
Jobname
The jobname of the assembly job
Stepname
The MVS stepname of the assembly step
Procstep
The MVS procedure step name of the assembly procedure step
The extra information shown on the More panel is the list of input files for the assembly. The information shown for each file is:

Figure 27. The Topic Help for the Job Id Tab

The right panel is the topic panel. It shows the current topic. If you invoke the help by pressing a **Help** button, the current topic relates to the button you pressed. For example, the figure shows the topic that results from pressing the **Job Id** tab **Help** button.

You can move to a different topic by clicking a hot link in a topic. A hot link is text that is underlined and colored. The cursor changes to a pointing finger when it is on a hot link.

The left panel has four tabs. They are:

Contents A structured table of contents. If you click on a topic heading, the associated topic is displayed in the topic panel. Double-click on a topic tagged with a book icon, and the underlying topics appear or disappear.

Index The index for the help. Find a topic by typing in the keyword. As you type, the highlight moves. Alternatively, scroll the index. When you double-click an index entry, the topic is displayed. If there are two or more topics for the entry, select a topic from the displayed list.

Search Type in a keyword, and click **List Topics**. Then select the topic you want from the resultant list. You can search for any word the help file.

Favorites A list of favorite topics. To add a topic to the list, display the topic in the topic area, then click **Add**. Topics are listed in alphabetic order.

To show just the topic panel, click the **Hide** icon.

To print a selected topic, or the selected topic and the following subtopics, click the **Print** icon.

You can change the font of the text in the topic panel from the **Internet Options** of the **Options** menu.

Using What's This help

To invoke What's This help, click on the question mark beside the **Close** box. A question mark is appended to the cursor. Then click on the item you want information about. A note is displayed. Clear the note by clicking again.

The OS/2 help

The OS/2 help system works within the standard OS/2 help framework.

You can look at a table of contents, find a topic by looking through the index, or search topics for a word.

Each individual topic is displayed in a window, and you can access a topic from the table of contents, index or search results, or by moving to the next topic.

ASMPUT messages

ASMP001S Unable to get system information. Return code *returncode***.**

Explanation: An operating system call failed when attempting to obtain information about the executable module.

User Response: Refer to your operating system documentation for information about the return code. Possibly your system is running short of some resource such as memory. Try closing down other applications, or reboot the system and try running ASMPUT again.

ASMP002S Cannot open ADATA file - *filename***.**

Explanation: The requested ADATA file *filename* cannot be found.

User Response: Enter a different file name, or else select the file from a directory listing.

ASMP003E Unable to find ADATA header at offset *fileOffset***.**

Explanation: A valid ADATA header record was expected, but not found. The ADATA file may be corrupt, or may contain invalid data.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP004I Skipping forward *charCount* **characters in ADATA file.**

Explanation: Data has been skipped in the ADATA file in attempt to find a valid header record.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP005S Unable to read ADATA file at offset *fileOffset***.**

Explanation: A read error occurred in the ADATA file at the specified offset.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP006S Unable to process architecture level *adataLevel* **at offset** *fileOffset***.**

Explanation: This version of ASMPUT only processes level 3 ADATA files. Either an old ADATA file is being used, or the file is corrupt.

User Response: Recreate the ADATA file and try again. Ensure that High Level Assembler Version 1.4 is being used to create the file, and that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP007S Unidentified record type *recordType* **at offset** *fileOffset***.**

Explanation: An unexpected record type was found in the ADATA file. This should not occur. The file may be corrupt.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP008S Bad ADATA record sequence at record *recordNumber***.**

Explanation: The ADATA records are not in the proper sequence. The file may be corrupt.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP010W Unable to load help file *helpFileName***.**

Explanation: The help file could not be found.

User Response: Check to see if the help file has been properly installed in the ASMPUT installation directory. If it is missing, try reinstalling ASMPUT.

ASMP012W ADATA file *adataFile* **is already loaded.**

Explanation: The specified ADATA file is already loaded.

User Response: Open another file.

ASMP013W Unassociated *recordType* **record at record number** *recordNumber***.**

Explanation: The specified record could not be associated with its proper parent type. There is a problem with the program that produced the ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP014W This program contains assembly diagnostics. The analysis may be invalid.

Explanation: The assembly contains diagnostics, and a correct analysis may not be possible.

User Response: To guarantee a valid analysis, remove assembly errors, rebuild a new ADATA file, and submit the new file to ASMPUT for analysis.

ASMP015S This does not appear to be a valid ADATA file.

Explanation: The file does not appear to be a valid ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP016S Execution directory appears to be invalid - *executableName*

Explanation: The directory from which ASMPUT is being executed appears to be invalid.

User Response: Examine the executableName and confirm that it conforms to the standard operating system rules for file naming. If it is invalid, reinstall ASMPUT in a new directory.

ASMP017E Missing External Symbol Dictionary record for ESDID *esdid* **at record number** *recordNumber***.**

Explanation: The specified record is missing. There is a problem with the program that produced the ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP018E Symbol redefinition has occurred at record number *recordNumber***.**

Explanation: A symbol has been defined twice. There is a problem with the program that produced the ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP019E Symbol cross reference to non-existent statement *statementNumber* **at record number** *recordNumber***.**

Explanation: A cross reference record refers to a non-existent statement. There is a problem with the program that produced the ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP020E Invalid machine instruction at statement number *statementNumber***.**

Explanation: The machine instruction record is malformed. Either the ADATA record is corrupt, or the assembler is outputting machine instruction records containing bad instructions.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP021E Attempt to change parent of *label* **from** *fromLabel* **to** *toLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP022E Unable to determine branch target.

Explanation: This is an internal flow analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP023E Multiple *recordType* **records found at record number** *recordNumber***.**

Explanation: Multiple records of the specified type have been found. This is an error in the ADATA file.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP024W Missing branch at end of flow block at statement number *statementNumber***.**

Explanation: Flow analysis has detected a code block which apparently ends without any exit point, such as a branch or a following code block.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP025W Unable to initialize the HTML Help system.

Explanation: The HTML Help system could not be initialized. It may not be correctly installed.

User Response: Ensure that the file HHCTRL.OCX exists in the Windows system directory. If necessary, re-install HTML Help. The most recent version can be found at http://msdn.microsoft.com/workshop/author/htmlhelp/.

ASMP026E Missing Library record for cross-reference record number *recordNumber***.**

Explanation: This is an error in the ADATA file structure.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP027E Missing Source record for cross-reference statement *statementNumber* **at record number** *recordNumber***.**

Explanation: This is an error in the ADATA file structure.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP028E Node *label* **is not a child of parent node** *parentLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP029E Node *label* **has no parent but has a sibling** *siblingLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP030S Internal node not found for program *programName***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP031S Node *label* **not found in hash table.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP032I End of search.

Explanation: A find request has reached the end of the source file. The requested text has not been found.

User Response: Try another find request if the required information has not been found.

ASMP033E A complex symbol expression has been specified on the END statement. This is not currently supported, so the program cannot be correctly analyzed.

Explanation: Only simple variable names are supported as entry points on the END statement. Complex expressions are permitted by the assembler, but are not currently supported by ASMPUT.

User Response: Change the expression on the END statement, if possible, and re-assemble the program.

ASMP034S Internal error. Source arc *arcLabel* **not linked to source node** *nodeLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP035S Internal error. Unlinked source arc *label* **has sibling links.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP036S Internal error. Target arc *arcLabel* **not linked to target node** *nodeLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP037S Internal error. Unlinked target arc *arcLabel* **has sibling links.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP038E ADATA files assembled with the XOBJECT option cannot be processed. Please re-assemble the program with the NOXOBJECT option.

Explanation: The XOBJECT option omits essential information from the ADATA file, so the analysis of the program cannot be performed using this option.

User Response: Re-assemble the program with the NOXOBJECT option.

ASMP039W *registerType* **Register** *registerNumber* **may be referenced before it has been set.**

Explanation: Flow analysis has detected that a register may be used by an instruction without previously being set by another instruction.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP040W Instruction contains a reference to an absolute memory address.

Explanation: The instruction is referencing a location in the first 4K of low memory.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP041W Instruction uses register *registerNumber* **as an index register but has no base register.**

Explanation: The instruction has coded a base register using the index register specification. This has no effect on the final result of the instruction.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP042E Unable to resolve the second operand address.

Explanation: This is an internal flow analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP043E Nominal value operand lengths in ADATA storage record are inconsistent.

Explanation: This is an internal flow analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP044W Instruction reference is not aligned to an operand boundary.

Explanation: The instruction is referencing an operand which is not properly aligned to the operand's size.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP045E Invalid ADATA Source record type *recordType* **at record number** *recordNumber***.**

Explanation: This is an error in the ADATA file structure.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP046E Invalid ADATA Source record *originType* **origin** *sourceType* **at record number** *recordNumber***.**

Explanation: This is an error in the ADATA file structure.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP047W Expected operand size (*instructionSize*) is greater than the referenced operand size (*operandSize*).

Explanation: The expected operand for this instruction is greater than the actual operand size (for example, a load instruction referencing a halfword operand). This may be valid if it is necessary to span multiple operands.

User Response: Examine the assembler source code and determine whether this is a genuine problem. If it is, correct the problem and re-assemble the code.

ASMP048W Code may be unreachable.

Explanation: Flow analysis was unable to reach this code section. This may be due to the inability of flow analysis to resolve an indexed branch table.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP049W Instruction is referencing code as data.

Explanation: An instruction is referencing assembled data as a code location, such as the target of a branch.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP050W *registerType* **Register** *registerNumber* **may not contain a valid address.**

Explanation: The instruction is using the contents of the specified register as a location address, but flow analysis has determined that the register may contain other data not representing a location address.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP051W A data statement has been detected within a code sequence.

Explanation: An assembler statement which generates data has been detected within the code stream.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP052W Serialization and checkpoint-synchronization function. Degraded performance may occur.

Explanation: This special form of the BCR instruction has been detected, and may lead to reduced performance if it has been used unnecessarily.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP053E Unable to find executable code at entry address *entryAddress***.**

Explanation: There is no executable code at the specified entry address.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP054E Unable to find original static flow record for statement *statementNumber***.**

Explanation: This is an internal flow analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP055E MarkAncestors found bottom node *label* **with missing AsmNode.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP056W Node layering cannot assign *nodeCount* **nodes to a layer. They are assigned to the highest layer.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP057W Duplicate external entry *entryLabel* **point detected.**

Explanation: Two external entry points with the same name have been detected.

User Response: Remove the ADATA file containing the duplicate entry point from the analysis.

ASMP058E Unable to find external reference entry for V-constant id *externalID***.**

Explanation: This is an error in the ADATA file structure.

User Response: Recreate the ADATA file and try again. Ensure that the version of the High Level Assembler being used to create the file is not higher than that of the ASMPUT. Also ensure that the recommended maintenance level as documented in the ASMPUT installation instructions has been applied. If the problem persists, report it to IBM service.

ASMP059W Source file compilation unit *unitNumber* **contains no code or data.**

Explanation: There is no code or data in the file.

User Response: Examine the assembler source code and determine whether this is a genuine problem, and if necessary correct the problem and re-assemble the code.

ASMP060E Internal error. Island/layer *label* **has only one child node** *childLabel***.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

ASMP061E Internal error. Bottom node *label* **has no island.**

Explanation: This is an internal graph analysis error. It should not occur.

User Response: Report the problem to IBM service.

Chapter 5. Using the Cross-Reference Facility

Introduction to ASMXREF . . . 106
Invoking the Cross-Reference Facility . . . 108
  Invoking ASMXREF on MVS . . . 109
    MVS JCL Example . . . 109
    Sample Procedures . . . 113
  Invoking ASMXREF on CMS . . . 115
    ASMXREF Control File . . . 116
    ASMXREF Token Statement File . . . 116
    ASMXREF Source List File . . . 117
    Default Options File . . . 117
    ASMXREF Language File . . . 118
    ASMXSCAN EXEC . . . 119
    ASMXRPT EXEC . . . 119
  Invoking ASMXREF on VSE . . . 120
    VSE JCL Example . . . 121
ASMXREF Control Statements . . . 125
  * . . . 125
  Library . . . 125
  Include . . . 126
  Exclude . . . 127
  Parm . . . 127
  Report . . . 128
ASMXREF Token Statement . . . 128
  Token . . . 129
  Scanning Rules for ASMXREF . . . 131
    Generic Matching Rules . . . 131
ASMXREF Options . . . 132
ASMXREF XRFLANG Statements . . . 132
  Default Token Segment . . . 133
  Language Segment . . . 134
ASMXREP Options . . . 135
Understanding the Reports . . . 135
  Languages Supported by Reports . . . 136
  Control Flow (CF) Report . . . 136
    C Family References . . . 137
    PL Family References . . . 137
    REXX References . . . 138
  Lines Of Code (LOC) Report . . . 139
    Changed Source Instruction (CSI) Measurements . . . 140
  The LOOC report . . . 145
    The LOC per Class section . . . 146
    The LOC per Object section . . . 146
    The Objects per Class section . . . 147
  Macro Where Used (MWU) Report . . . 147
  Spreadsheet Oriented Report (SOR) . . . 148
    File Transfer to PC . . . 149
  Symbol Where Used (SWU) Report . . . 150
  Token Where Used (TWU) Report . . . 154
  Tagged Source Program (TSP) . . . 155
ASMXREF Messages . . . 158

© Copyright IBM Corp. 1992, 2004

Message List 159
ASMXREF User Abends 169

Introduction to ASMXREF

The Cross-Reference Facility (ASMXREF) is a flexible source code cross-referencing tool to help you determine variable and macro usage, and to locate specific uses of arbitrary strings of characters. ASMXREF reads libraries for symbols, macros, and tokens and generates reports to help you evaluate the search results.

As well as its value in maintaining applications, ASMXREF helps you quickly identify selected fields of interest.

ASMXREF provides token scanning facilities for source code in the languages such as:

- Assembler
- C
- C++
- COBOL
- FORTRAN
- PL/I
- REXX

For details on all the languages supported by each report see Table 19 on page 136.

ASMXREF saves any informational or error messages in a message file (for details see “ASMXREF Messages” on page 158).

You can use ASMXREF to generate the following reports:

Control Flow (CF)

The CF report tabulates all intermodule program references as a function of member or entry point name. It can list references either in the order of the members *referring to* the subject entry point or the entry point names *referred by* the subject member, depending on the sort order.

For each part processed, the CF report can handle up to 256 internal procedure names and 1024 entry point names.

Reference names that exceed 64 characters are truncated.

ASMXREF classifies each reference by type. The classification is language specific. For details see “Control Flow (CF) Report” on page 136.

Lines of Code (LOC)

Provides a count, arranged by part and by component, of the number of source lines and comments in the part, and the shipped source instructions (SSI), which are the number of instructions within each part scanned, both executable and non-executable, that are not spaces or comments. As well, the report shows the changed source instructions (CSI), which are the number of unique SSI that have been modified in each part categorized by added, changed, deleted, moved, and so on.

In addition, the LOC Report provides a summary report of CSI arranged by programmer.

Lines of OO Code (LOOC)
Provides, for C++, the Lines of Code (LOC) per Class and per Object, and Objects per Class.

Macro Where Used (MWU)
Identifies calls to all macros, functions invoked, and all copy books copied and included. The report includes the type and frequency of the use of the macro, or function, and the reference. This report identifies external referenced entities. These entities can be subroutine calls, macro invocations or the inclusion of copy books. For details see "Macro Where Used (MWU) Report" on page 147.

Spreadsheet Oriented (SOR)
Shows occurrences of tokens in the search library. This report processes the default set of tokens, provided with ASMXREF, which contains useful fields of interest, such as `DATE`, and `YY/MM/DD`. You can supplement the default tokens with your own tokens, or turn off processing of the defaults and replace them with your own token list. You can specify tokens that ASMXREF is to include in the search generically (with wildcards), or explicitly (with the exact characters ASMXREF is to include in the search). You must specify exclude tokens explicitly. The report is in a comma-delimited format that you can import into a spreadsheet application such as Lotus 1-2-3.

When you run the ASMXREF scan phase for the TWU and SOR reports, ASMXREF generates the Tagged Source Program (TSP). The ASMXREF report phase uses the TSP to create the TWU and SOR reports. The TSP contains the original source code interspersed with ASMXREF generated comment records in the syntax of the language scanned. These comment records contain both the token string encountered and a cumulative count of the number of times ASMXREF has found the token so far in the source file. For details on the Spreadsheet Oriented report see "Spreadsheet Oriented Report (SOR)" on page 148. For details on the TSP see "Tagged Source Program (TSP)" on page 155.

Symbol Where Used (SWU)
Lists all the symbols (variables or macros) used in the source code and the type of reference to each symbol. For details see "Symbol Where Used (SWU) Report" on page 150.

Token Where Used (TWU)
Shows occurrences of tokens in the search library. This report processes the default set of tokens, provided with ASMXREF, which contains useful fields of interest, such as `DATE`, and `YY/MM/DD`. You can supplement the default tokens with your own tokens, or turn off processing of the defaults, and replace them with your own token list. You can specify tokens that ASMXREF is to include in the search either generically (with wildcards), or explicitly (using the exact characters ASMXREF is to include in the search). You must specify exclude tokens explicitly.

When you run the ASMXREF scan phase for the TWU and SOR reports, ASMXREF generates the Tagged Source Program (TSP). The

ASMXREF report phase uses the TSP to create the TWU and SOR reports. The TSP contains the original source code interspersed with ASMXREF generated comment records in the syntax of the language scanned. These comment records contain both the token string encountered and a cumulative count of the number of times ASMXREF has found the tokens so far in the source file. For details on the TWU report see “Token Where Used (TWU) Report” on page 154. For details on the TSP report see “Tagged Source Program (TSP)” on page 155.

Invoking the Cross-Reference Facility

ASMXREF runs in two phases:

Scan The scan phase extracts information from the specified library to create intermediate data files. During the scan phase ASMXREF uses:

- A control file: on MVS, in or identified with the SYSIN DD statement; on VSE, in SYSIPT or identified with an ASSIGN SYSIPT statement; on CMS, a file with a filetype of CNTL. The control file contains statements that specify the library to scan, the source files to include or exclude, and the reports to create. For details on the control statements see “ASMXREF Control Statements” on page 125.
- A token statement file (XRFTOKN) that contains the tokens you have specified for the TWU and SOR reports. For details on the token statements, see “ASMXREF Token Statement” on page 128.
- A language and default token file (XRFLANG) that contains:
  - The languages supported by ASMXREF.
  - The language-specific verbs excluded from the ASMXREF scan phase.
  - The default token statements used by the TWU and SOR reports. You can turn off the processing of the default tokens with the TOKEN NODEFLT statement, as described in “ASMXREF Token Statement” on page 128.

  For a description of the XRFLANG file, see “ASMXREF XRFLANG Statements” on page 132.

ASMXREF scans the requested source files and writes the necessary data to an intermediate data file for each requested report.

Report The report phase uses the intermediate data files to generate each report.

ASMXREF runs on MVS, CMS, and VSE. The following sections describe how to invoke ASMXREF on each of these platforms.

Invoking ASMXREF on MVS

On MVS, you invoke ASMXREF as a batch program using Job Control Language (JCL). The following MVS files are supplied with ASMXREF:

Table 15. MVS Files Supplied with ASMXREF

Filename	Contents
ASMXRUN	Sample MVS JCL that invokes the supplied cataloged procedures.
ASMXSCAN	A cataloged procedure that runs the program ASMXREF.
ASMXRPT	A cataloged procedure that runs the program ASMXREP.
XRFLANG	A sample XRFLANG file containing: • The languages supported by ASMXREF and a sample of language-specific exclude verbs. • The default tokens.
XRFTOKN	A sample XRFTOKN file containing a comment record.
ASMXREF	A program that scans the specified libraries and generates intermediate data files.
ASMXREP	A program that reads the intermediate data files and creates the required reports.

When creating the TWU and SOR reports, ASMXREF searches the source files for the default tokens specified in the XRFLANG file and for any tokens you have specified in the XRFTOKN file. If you need your own tokens make a copy of the sample token statement (XRFTOKN) file supplied with ASMXREF. Enter the token statements you need, one per line.

If you have write access to the XRFLANG file you can modify the default tokens in this file. You can also change, or add, the verbs under the language segment header. Generally, the XRFLANG file is modified to suit your environment after installation and the file need not change. Add any additional tokens that you require to your XRFTOKN file. For details on customizing the XRFLANG file see “ASMXREF XRFLANG Statements” on page 132.

For a description of the format of the token statements see “ASMXREF Token Statement” on page 128.

The following sections describe how to run the supplied procedures.

MVS JCL Example

The simplified MVS JCL in Figure 28 on page 110 shows how to create the CF, LOC, LOOC, MWU, SWU, SOR and TWU reports. Before running this example, edit the lines highlighted by numbers (such as **1**) as described in the instructions following the example listing. For a full listing of the procedures supplied with ASMXREF, see “Sample Procedures” on page 113.

Note: ASMXREF dynamically allocates data sets, therefore you do not need to allocate DD statements.

```
//ASMXRUN   JOB <JOB CARD PARAMETERS>                                    1
//
//*********************************************************************
//*                                                                   *
//*   Licensed Materials - Property of IBM                            *
//*                                                                   *
//*   5692-234                                                        *
//*                                                                   *
//*   (C) Copyright IBM Corp. 1992, 2004. All Rights Reserved.        *
//*                                                                   *
//*   US Government Users Restricted Rights - Use,                    *
//*   duplication or disclosure restricted by GSA ADP                 *
//*   Schedule Contract for IBM Corp.                                 *
//*                                                                   *
//*********************************************************************
//*                                                                   *
//* ASMXRUN  JOB                                                      *
//*                                                                   *
//* THIS SAMPLE JCL WILL INVOKE THE ASMXSCAN AND ASMXRPT PROCEDURES.  *
//*                                                                   *
//*  CAUTION: THIS IS NEITHER A JCL PROCEDURE NOR A COMPLETE JOB.     *
//*  BEFORE USING THIS JOB, YOU WILL HAVE TO MAKE THE FOLLOWING       *
//*  MODIFICATIONS:                                                   *
//*                                                                   *
//* 1. CHANGE THE JOB CARD TO MEET YOUR SYSTEM REQUIREMENTS           *
//* 2. CHANGE #jcllib TO BE THE NAME OF THE USER JCL LIBRARY DATASET. *
//* 3. CHANGE #user TO BE THE USER NAME                               *
//* 4. CHANGE #user.source TO BE THE SOURCE LIBRARY TO SCAN           *
//* 5. CHANGE #source.name TO BE THE SOURCE MEMBER NAME.              *
//* 6. CHANGE #lang TO BE THE LANGUAGE OF THE SOURCE MEMBER           *
//*    (E.G. ASM FOR ASSEMBLER SOURCE)                                *
//*                                                                   *
//*********************************************************************
//* NOTE: UNCOMMENT THE FOLLOWING STATEMENT IF THE ASMXREF AND        *
//*       ASMXRPT PROCEDURES ARE PLACED IN YOUR USER JCL LIBRARY      *
//*       #jcllib RATHER THEN THE SYSTEM PROCEDURE LIBRARIES.         *
//*********************************************************************
//*JCL JCLLIB ORDER=(#jcllib)                                           2
//*
//*********************************************************************
//* STEP 1 CREATE INTERMEDIATE FILE                                   *
//*********************************************************************
//STEP1  EXEC ASMXSCAN,PARM.ASMXREF='NODUP',USER=#user,                 3
//       ASMPRFX=#hlq
//SYSIN  DD   *                                                        4
* SAMPLE CONTROL FILE FOR XREF
*
  LIBRARY  LIB=#user.source,TYPE=PDS
     INCLUDE MOD=#source.name,LANGUAGE=#lang
*
  REPORT   REPORT=CF                          CONTROL FLOW
  REPORT   REPORT=LOC                         LINES OF CODE
  REPORT   REPORT=LOOOC                       LINES OF OO CODE
  REPORT   REPORT=MWU                         WHERE/WHAT USED
  REPORT   REPORT=SOR                         SPREAD SHEET ORIENTED
  REPORT   REPORT=SWU                         SYMBOL WHERE USED
  REPORT   REPORT=TWU                         TOKEN WHERE USED
/*
//*********************************************************************
//* STEP 1A  DELETION OF INTERMEDIATE FILE IN CASE STEP1 FAILS.       *
//*          THIS WILL ALLOW THE JOB TO BE RERUN WITHOUT MANUAL       *
//*          DELETION OF * A DUPLICATE DATASET.                       *
```

Figure 28 (Part 1 of 2). Sample MVS ASMXREF JCL

```
//*******************************************************************
//DEL EXEC PGM=IDCAMS,COND=(0,EQ)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  DELETE #user.TWU.TAGGED.FILE                                      5
/*
//*******************************************************************
//* STEP 2   THIS PRODUCES THE CONTROL FLOW REPORT.                 *
//*******************************************************************
//STEP2 EXEC ASMXRPT,REPORT=CF,RECLEN=143,BLKSIZE=1430,USER=#user,
//          RPARM='60 MAC',COND=(0,NE),ASMPRFX=#hlq
//SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 3   THIS PRODUCES THE LINES OF CODE REPORT.                *
//*******************************************************************
//STEP3 EXEC ASMXRPT,REPORT=LOC,RECLEN=145,BLKSIZE=1450,USER=#user,
//          RPARM='60 MOD',COND=(0,NE),ASMPRFX=#hlq
//SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 4   THIS PRODUCES THE LINES OF OO CODE REPORT.             *
//* NOTE: UNCOMMENT THE FOLLOWING STATEMENTS IF THE LOOC REPORT IS   *
//*       REQUIRED. THIS IS AVAILABLE FOR THE CPP LANGUAGE.         *
//*******************************************************************
//*STEP4 EXEC ASMXRPT,REPORT=LOOC,RECLEN=99,BLKSIZE=990,USER=#user,
//*          RPARM='60',COND=(0,NE),ASMPRFX=#hlq
//*SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 5   THIS PRODUCES THE MODULE WHERE USED (MWU) REPORT        *
//*******************************************************************
//STEP5 EXEC ASMXRPT,REPORT=MWU,RECLEN=96,BLKSIZE=3936,USER=#user,
//          RPARM='60 MAC',COND=(0,NE),ASMPRFX=#hlq
//SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 6   THIS PRODUCES THE SYMBOL WHERE USED (SWU) REPORT        *
//*******************************************************************
//STEP6 EXEC ASMXRPT,REPORT=SWU,RECLEN=93,BLKSIZE=3999,USER=#user,
//           RPARM='60 SYM',COND=(0,NE),ASMPRFX=#hlq
//SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 7   THIS PRODUCES THE TOKEN WHERE USED (TWU) REPORT         *
//*          NOTE: THE LAST STEP TO REFERENCE THE SYSINDS DATASET    *
//*          FOR THE TWU OR SOR REPORT SHOULD SPECIFY                *
//*          DISP=(OLD,DELETE)                                       *
//*******************************************************************
//STEP7 EXEC ASMXRPT,REPORT=TWU,RECLEN=80,BLKSIZE=80,USER=#user,    6
//           RPARM='',COND=(0,NE),ASMPRFX=#hlq
//SYSINDS DD DSN=*.STEP1.ASMXREF.XRFTWU,DISP=(OLD,KEEP)
//SYSINOU DD SYSOUT=*
//*
//*******************************************************************
//* STEP 8   THIS PRODUCES THE SPREADSHEET ORIENTED REPORT (SOR).    *
//*          NOTE: THE LAST STEP TO REFERENCE THE SYSINDS DATASET    *
//*          FOR THE TWU OR SOR REPORT SHOULD SPECIFY                *
//*          DISP=(OLD,DELETE) IF YOU DON NOT WISH TO KEEP THE       *
//*          TASF FILE.                                              *
//*******************************************************************
//STEP8 EXEC ASMXRPT,REPORT=SOR,RECLEN=80,BLKSIZE=80,USER=#user
//           RPARM=', ''',COND=(0,NE),ASMPRFX=#hlq
//SYSINDS DD DSN=*.STEP1.ASMXREF.XRFTWU,DISP=(OLD,DELETE)
//*******************************************************************
//* NOTE: COMMENT THE FOLLOWING STATEMENT IF THE USER DOES NOT       *
//*       REQUIRE A SPREADSHEET DATASET TO BE CREATED.              *
//*******************************************************************
//SYSINOU DD SYSOUT=*
//
```

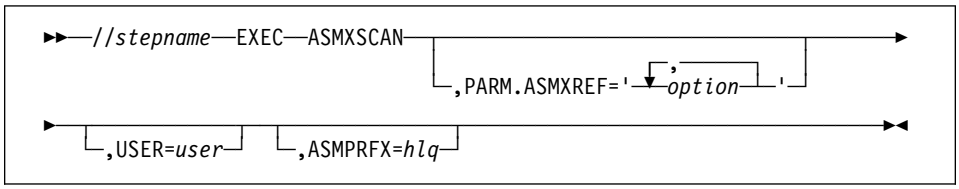
Figure 28 (Part 2 of 2). Sample MVS ASMXREF JCL

1 Add the job parameters tomeet your system requirements.

2 If you store the ASMXSCAN or ASMXRPT procedures in the JCL library rather than the system procedure library, remove the comment characters on this line, and then replace `#jcllib` with the data set name of the JCL library.

3 Replace `NODUP` with the options you need for the ASMXREF run, and replace `#user` with your userid.

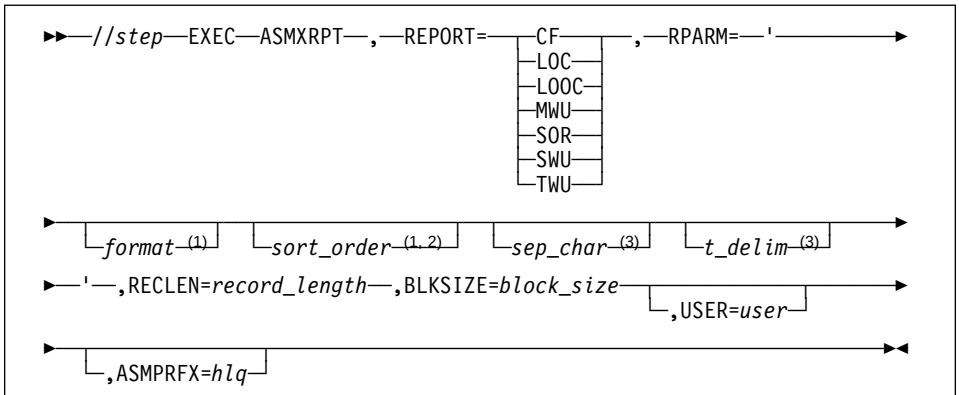
EXEC ASMXSCAN runs the procedure ASMXSCAN, which runs the program ASMXREF. ASMXREF requires that you specify at least one option with the PARM parameter. For details of the options available with ASMXREF see "ASMXREF Options" on page 132. The format of the ASMXREF statement is:



4 Enter the ASMXREF control statements you need, either immediately following the `SYSIN DD *` statement, as in the sample JCL shown in Figure 28 on page 110, or enter the name of the data set that contains the control statements. For details on the control statements, see "ASMXREF Control Statements" on page 125.

5 Replace `#user` with your userid.

6 Replace `#user` with your userid and specify any options that you need for each report in RPARM. Run this procedure for each report that you need. The format of the statement is:



Notes:

1 These options are not available for the SOR report.

2 This option not required for the LOOC report.

3 These options are available only for the SOR report.

For details of the reports available see "Understanding the Reports" on page 135. For details of the options available, see "ASMXREP Options" on page 135. You must specify only one report with this statement.

Note: All the parameters are positional. You must enter them in the order shown above, or you can enter just the REP parameter and leave the other parameters blank. If you enter the parameters in the wrong order ASMXREF issues an error message.

Sample Procedures

The following figures show the two sample procedures supplied with ASMXREF. After installing ASMXREF your systems programmer must copy these procedures into your procedure library and modify them to suit your environment. Once modified, use the sample JCL shown in Figure 28 on page 110 to invoke the procedures. If the names of the procedures have changed, modify the JCL to reflect the change.

```
//********************************************************************
//*   Licensed Materials - Property of IBM                           *
//*                                                                  *
//*   5696-234   5647-A01                                            *
//*                                                                  *
//*   (C) Copyright IBM Corp. 1992, 2004. All Rights Reserved.       *
//*                                                                  *
//*   US Government Users Restricted Rights - Use,                   *
//*   duplication or disclosure restricted by GSA ADP                *
//*   Schedule Contract with IBM Corp.                               *
//*                                                                  *
//********************************************************************
//********************************************************************
//*                                                                  *
//* ASMXSCAN PROC                                                    *
//*                                                                  *
//* THIS SAMPLE JCL PROC IS INVOKED FROM THE ASMXRUN SAMPLE JCL.     *
//* IT INVOKES THE ASMXREF PROGRAM.                                  *
//*                                                                  *
//*                                                                  *
//********************************************************************
//ASMXSCAN PROC SYSOUT='*',
//            USER=USER,
//            ASMPRFX=HLA
//ASMXREF  EXEC PGM=ASMXREF,REGION=4M
//*
//STEPLIB  DD DISP=SHR,DSN=&ASMPRFX..SASMMOD2
//*
//XRFCF    DD DSN=&&CF,           DATA FOR CONTROL FLOW REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,10),RLSE)
//XRFLOC   DD DSN=&&LOC,          DATA FOR LINES OF CODE REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,10),RLSE)
//XRFLOOC  DD DSN=&&LOOC,         DATA FOR LINES OF OO CODE REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,10),RLSE)
//XRFMWU   DD DSN=&&MWU,          DATA FOR MACRO WHERE USED REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,10),RLSE)
//XRFSWU   DD DSN=&&SWU,          DATA FOR SYMBOL WHERE USED REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,20),RLSE)
//XRFSWUO  DD DSN=&&SWUO,         DATA FOR SYMBOL WHERE USED REPORT
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,20),RLSE)
//XRFMDLOG DD DSN=&&MDLO,
//            DISP=(NEW,PASS),UNIT=SYSALLDA,SPACE=(TRK,(0,20),RLSE)
//XRFTWU   DD DSN=&USER..TWU.TAGGED.FILE,
//            DCB=(LRECL=80,BLKSIZE=3200,DSORG=PS),
//            SPACE=(CYL,(5,1),RLSE),
//            UNIT=SYSALLDA,
//            DISP=(NEW,CATLG)
//XRFSYMLB DD DUMMY
//XRFSCIP  DD DUMMY
//XRFTST   DD DUMMY
//SYSPRINT DD SYSOUT=&SYSOUT
//XRFTOKN  DD DISP=SHR,DSN=&ASMPRFX..SASMSAM2(ASMXTOKN)     1
//XRFLANG  DD DISP=SHR,DSN=&ASMPRFX..SASMSAM2(ASMXLANG)     2
//SYSIN    DD DSN=NULLFILE,DISP=SHR
```

Figure 29. Sample ASMXSCAN Procedure

1 You require this DD statement only for the TWU and SOR reports if you need to supplement, or replace, the default tokens with your own tokens. Replace &ASMPRFX..SASMSAM2(ASMXTOKN) with the name of the data set containing your token statements.

If the default tokens are sufficient, and additional tokens are not required, replace DISP=SHR,DSN=&ASMPRFX..SASMSAM2(ASMXTOKN) with DUMMY. For example:

```
//XRFTOKN DD DUMMY
```

For details on the token statements see “ASMXREF Token Statement” on page 128.

If you are not creating the TWU or SOR reports remove this statement.

2 Replace &ASMPRFX..SASMSAM2(ASMXLANG) with the name of the data set containing the XRFLANG file. For details of the XRFLANG file see “ASMXREF XRFLANG Statements” on page 132.

```
//*********************************************************************
//*   Licensed Materials - Property of IBM                            *
//*                                                                   *
//*   5696-234   5647-A01                                             *
//*                                                                   *
//*   (C) Copyright IBM Corp. 1992, 2004. All Rights Reserved.        *
//*                                                                   *
//*   US Government Users Restricted Rights - Use,                    *
//*   duplication or disclosure restricted by GSA ADP                 *
//*   Schedule Contract with IBM Corp.                                *
//*                                                                   *
//*********************************************************************
//*********************************************************************
//*                                                                   *
//* ASMXRPT PROC                                                      *
//*                                                                   *
//* THIS SAMPLE JCL PROC IS INVOKED FROM THE ASMXRUN SAMPLE JCL.      *
//* IT INVOKES THE ASMXRPT PROGRAM.                                   *
//*                                                                   *
//*                                                                   *
//*********************************************************************
//ASMXRPT PROC REPORT=,      A VALID 2-3 LETTER REPORT ACRONYM
//            RPARM=,       REPORT PARAMETERS
//            RECLEN=,      RECORD LENGTH
//            BLKSIZE=,     BLOCK SIZE
//            SYSOUT='*',   SYSOUT CLASS
//            SPACE=5,      REPORT SPACE IN TRACKS
//            USER=USER,    USER ID OR HIGH LEVEL QUALIFIER
//            ASMPRFX=HLA   HIGH LEVEL QUALIFIER FOR TOOLKIT LIBRARY
//*
//ASMXRPT EXEC PGM=ASMXREP,REGION=3M,
//            PARM=('&REPORT &RPARM ')
//STEPLIB  DD DISP=SHR,DSN=&ASMPRFX..SASMMOD2
//SYSPRINT DD SYSOUT=&SYSOUT
//SYSINDS DD DSN=&&&REPORT,DISP=(OLD,PASS)
//SYSINOU DD DSN=&USER..XREFOUT.&REPORT,
//            DISP=(NEW,CATLG),UNIT=SYSALLDA,
//            DCB=(RECFM=FBA,LRECL=&RECLEN,BLKSIZE=&BLKSIZE),
//            SPACE=(TRK,(&SPACE,5),RLSE)
```

Figure 30. Sample ASMXRPT Procedure

Invoking ASMXREF on CMS

On CMS you invoke ASMXREF with REXX EXECs. ASMXREF is supplied with the following CMS files:

Table 16. CMS Files Supplied with ASMXREF

Filename	Filetype	Contents
ASMXSCAN	EXEC	A REXX EXEC that runs the program ASMXREF.
ASMXRPT	EXEC	A REXX EXEC that runs the program ASMXREP.
ASMXREF	MODULE	A program that scans the specified libraries and creates intermediate data files.
ASMXREP	MODULE	A program that reads the intermediate data files and creates the required reports.
ASMTEST	CNTL	A sample control file.
ASMTEST	DATATOKN	A sample token statement (XRFTOKN) file.
ASMTEST	EXEC	A sample source list file.
ASMTEST	DEFAULTS	Contains the default ASMXREF options.
ASMTEST	DATALANG	A sample XRFLANG file that contains: • The languages supported by ASMXREF and a sample of language-specific exclude verbs. • The default tokens used by the TWU and SOR reports.
ASMXSEP	EXEC	A REXX EXEC that splits the TSP into its component files. For a description of splitting the TSP see "Tagged Source Program (TSP)" on page 155.

Take the following steps to run ASMXREF:

1. Before running ASMXREF make a copy of the following information files:
   - ASMTEST CNTL, the sample control file
   - ASMTEST DATATOKN, the sample token statement (XRFTOKN) file (used only with the TWU and SOR reports where it is optional)
   - ASMTEST EXEC, the sample source list file
   - ASMTEST DEFAULTS, the default options file
   - ASMTEST DATALANG, the sample language (XRFLANG) file
2. Modify the details in each of these files to suit your ASMXREF run.
3. Save each of the files with a new filename. They must all have the same filename, but retain the existing filetype. For example:

   ```
   MYXREF CNTL
   MYXREF DATATOKN
   MYXREF EXEC
   MYXREF DEFAULTS
   MYXREF DATALANG
   ```

   Refer to the following sections for details of the information required in each of these files.

4. Remove the comment characters from the appropriate FILEDEF XRFTOKN statement in the ASMXSCAN EXEC. For details of this statement see “ASMXREF Token Statement File” on page 116.
5. When the information files have been created, run the ASMXSCAN EXEC to create the intermediate data files.
6. Run the ASMXRPT EXEC to create each report that you need.

   For details on the format of the EXEC commands and details of the files created by these EXECs, see “ASMXSCAN EXEC” on page 119 and “ASMXRPT EXEC” on page 119.

ASMXREF Control File

The control file contains the control statements for your ASMXREF run.

Make a copy of the sample control file supplied with ASMXREF and enter the control statements you need for your ASMXREF run. Enter each control statement on a separate line, starting in column one. You can give the file any valid filename, but the filetype must be CNTL.

For an example of a control file, named ASMTEST CNTL A, see Figure 31.

```
*  Sample control file for ASMXREF
*
LIBRARY  LIB=ASMTEST,TYPE=CMS,LANGUAGE=ASM
*
REPORT   REPORT=CF                                  CONTROL FLOW
REPORT   REPORT=LOC                                 LINES OF CODE
REPORT   REPORT=LOOC                                LINES OF OO CODE
REPORT   REPORT=TWU                                 TOKEN  WHERE USED
REPORT   REPORT=MWU                                 MACRO  WHERE USED
REPORT   REPORT=SWU                                 SYMBOL WHERE USED
```

Figure 31. Example Control File for CMS ASMXREF EXEC

For full details of the control statements you can use with ASMXREF see “ASMXREF Control Statements” on page 125.

ASMXREF Token Statement File

The token statement (XRFTOKN) file contains your token statements for the ASMXREF run.

Use this file only when creating the TWU or SOR reports and you need to specify tokens in addition to the default tokens in the XRFLANG file. When ASMXREF runs the scan phase it searches for the default tokens specified in the XRFLANG file and for any additional tokens you have specified in the XRFTOKN file.

Check the XRFTOKN file definition statements in ASMXSCAN EXEC:

- If the default tokens are sufficient and additional tokens are not required, or when running reports other than the TWU or SOR, modify the file definition statements as follows:

  ```
  IssueFileDefs:
   /*'FILEDEF XRFTOKN  DISK' FileName 'DATATOKN' WorkMode */
     'FILEDEF XRFTOKN  DUMMY'
  ```

- If you require tokens in the XRFTOKN file modify the file definition statements as follows:

```
IssueFileDefs:
'FILEDEF XRFTOKN  DISK' FileName 'DATATOKN' WorkMode
/*'FILEDEF XRFTOKN  DUMMY' */
```

When you need to supplement or replace the default tokens with your own tokens, make a copy of the sample token (XRFTOKN) file supplied with ASMXREF. Enter the token statements you need, one per line. For details of the default tokens see “ASMXREF XRFLANG Statements” on page 132. For details of the token statements you can use see “ASMXREF Token Statement” on page 128.

Save your token file with the same filename as the control file and a filetype of DATATOKN.

Figure 32 shows an example of a token statement in a file named ASMTEST DATATOKN A.

```
*  Sample token statement file for ASMXREF
TOKEN INC='ABC'
```

Figure 32. Example Token Statement File for CMS ASMXREF EXEC

ASMXREF Source List File

Contains the names of the source files that ASMXREF scans.

Make a copy of the sample source list file supplied with ASMXREF and enter the filenames of the files you need ASMXREF to scan. Give the source list file a filetype of EXEC, with any filename you choose, but for consistency it is advisable to give it the same name as the control file. You must specify this filename on the LIB parameter of the LIBRARY control statement. For details, see “ASMXREF Control Statements” on page 125.

The format of each line in the source list file is:

```
&1 &2 filename filetype filemode
```

where *filename*, *filetype*, and *filemode* identify a source file to be scanned. The following example scans files with the filename ASMSRC and a filetype of ASSEM on any filemode:

```
&1 &2 ASMSRC ASSEM  *
```

All source files must be written in the same language for each ASMXREF run.

Default Options File

Contains the default options for ASMXREF and ASMXREP.

Make a copy of this file; give it the same filename as the control file and a filetype of DEFAULTS.

You can change some of the default options in this file, but ASMXREF does not allow you to change others. Comment lines in the file indicate which options you can change.

If you need to modify any of these options change them to suit your ASMXREF run, and save the file. For details of the options available see “ASMXREF Options” on page 132 and “ASMXREP Options” on page 135.

```
* ********************************************************** *
*   Licensed Materials - Property of IBM                     *
*                                                            *
*   5696-234                                                 *
*                                                            *
*   (C) Copyright IBM Corp. 1975, 2004. All Rights Reserved. *
*                                                            *
*   US Government Users Restricted Rights - Use,             *
*   duplication or disclosure restricted by GSA ADP          *
*   Schedule Contract with IBM Corp.                         *
*                                                            *
* ********************************************************** *
* ---------------------------------------------------------------*
* This file contains Local Defaults for ASMXSCAN and ASMXRPT EXEC  *
* procedures.                                                    *
* Each assignment statement must remain in proper REXX format.   *
*                                                                *
* Any line beginning with an asterisk, is a comment and ignored. *
*                                                                *
* Please read the instructions preceding each group of parameters. *
*                                                                *
* ASMXSCAN and ASMXRPT will use the ASMTEST DEFAULTS file found in *
* the normal CMS search sequence.                                *
* Note:  ASMTEST DEFAULTS is the supplied defaults file.         *
* ---------------------------------------------------------------*
*
* ---------------------------------------------------------------*
* The following defaults cannot be changed.                      *
* ---------------------------------------------------------------*
DefaultASMXModule       = 'ASMXREF'
DefaultReportModule     = 'ASMXREP'
DefaultOpcodeTable      = 'ASMOP370'
* ---------------------------------------------------------------*
* The following Defaults MAY be changed.                         *
* ---------------------------------------------------------------*
DefaultDuplicates       = 'DUP'
DefaultMessageLevel     = '4'
DefaultPageLength       = '60'
DefaultWorkMode         = 'A'
DefaultCNTLMode         = '*'
DefaultReturnMsg        = 'NO'
*
```

Figure 33. Default Options File for ASMXREF EXEC

ASMXREF Language File

Contains a list of the languages supported by ASMXREF and exclude verbs (words) specific to the language. ASMXREF excludes the verbs (words) when it scans a source file in the specified language. The file also contains the default token statements.

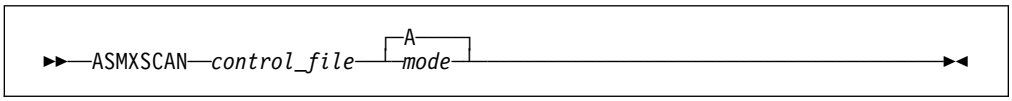
Make a copy of this file; give it the same filename as the control file and a filetype of DATALANG.

You can modify, or add to, the language-specific exclude words and the default token list. ASMXREF only supports the languages supplied in the file, therefore you cannot add any other language.

For details on the default tokens supplied with ASMXREF and the supported languages see “ASMXREF XRFLANG Statements” on page 132.

ASMXSCAN EXEC

The ASMXSCAN EXEC runs the scan phase. This searches the source files listed in the source list file and creates intermediate data files named *filename* DATA*rpt* A, where *filename* is the name of the control file and *rpt* is the report acronym. ASMXSCAN uses the options in the *filename* DEFAULTS A file, where *filename* is the same name as the control file. The format of the ASMXSCAN EXEC command is as follows:



where *control_file* is the name of the control file.

For example:

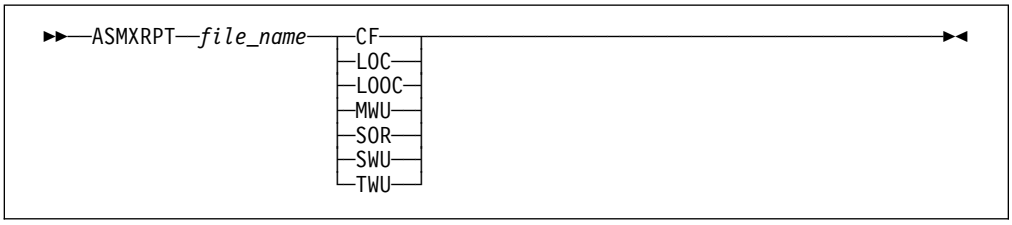
```
ASMXSCAN ASMTEST A
```

The previous example searches the source files listed in the file ASMTEST EXEC A, using the control statements stored in the control file ASMTEST CNTL A, and generates an intermediate data file for each of the required reports.

When using the control file in Figure 31 on page 116 ASMXREF creates output files named ASMTEST DATACF, ASMTEST DATALOC, ASMTEST DATATWU, ASMTEST DATAMWU, and ASMTEST DATASWU. The file ASMTEST DATATWU contains the TSP for both the TWU and SOR reports.

ASMXRPT EXEC

The ASMXRPT EXEC runs the ASMXREF report phase, creating an output file for the requested report. The name of this file is *filename* OUT*rpt* A, where *filename* is the same filename as the control file and *rpt* is the report acronym. You can only specify one report each time you run the EXEC, so you must run the EXEC for every report required. The format of the command is as follows:



where *filename* is the name of the scan phase control file.

In the following example ASMXRPT uses the input file named ASMTEST DATATWU A, and creates the TWU report in a file named ASMTEST OUTTWU A.

```
ASMXRPT ASMTEST TWU
```

Invoking ASMXREF on VSE

On VSE, you invoke ASMXREF as a batch program. ASMXREF is supplied with the following VSE files:

Table 17. VSE Files Supplied with ASMXREF

Filename	Contents
ASMXRUN	A sample VSE JCL job which invokes the supplied programs.
ASMXREF	A program that scans specified libraries and generates intermediate data files.
ASMXREP	A program that reads the intermediate data files and creates the required report.
XRFLANG	A sample XRFLANG file containing: • The languages supported by ASMXREF and a sample of language-specific exclude verbs. • The default tokens.
XRFTOKN	A sample token statement file containing a comment statement.
ASMXJC2S	JCL which catalogs the XRFLANG and XRFTOKN file names in the VSAM catalog.

When creating the TWU and SOR reports, ASMXREF searches the source files for the default tokens specified in the XRFLANG file and for any tokens you have specified in the XRFTOKN file. If you need your own tokens make a copy of the sample token statement (XRFTOKN) file supplied with ASMXREF. Enter the token statements you need, one per line. If you do not need to supplement the default tokens, enter just a comment line in the XRFTOKN file.

If you have write access to the XRFLANG file you can modify the default tokens in this file. You can also change, or add, the verbs under the language segment header in this file. Generally, the XRFLANG file is modified to suit your environment after installation and the file need not change. Add any additional tokens that you require to your XRFTOKN file.

For details of the format of the token statement see “ASMXREF Token Statement” on page 128. For details of the default tokens supplied with ASMXREF see “ASMXREF XRFLANG Statements” on page 132.

The following sections describe the job control statements required to run these programs.

Note: After installing ASMXREF, or if you change the XRFLANG or XRFTOKN files, you must copy the contents of the files into VSAM managed SAM files. ASMXREF is supplied with a sample JCL job, named ASMXJC2S, that runs this job. This JCL copies the contents from the librarian members (using '* $$ SLI') into VSAM managed SAM files. if you rerun this job change the DLBL statements from DISP=(NEW,KEEP) to DISP=(OLD,KEEP).

VSE JCL Example

The simplified JCL in Figure 34 shows how to create the CF, LOC, LOOC, MWU, SWU, TWU and SOR reports. Before running this example edit the lines highlighted by numbers (such as **1**) as described in the instructions following Figure 34.

```
@ $$ JOB JNM=ASMXRUN,LDEST=(*,USERID),CLASS=0                              1
// JOB GSCAN
* ---------------------------------------------------------------------
* NOTE 1: PLEASE CHANGE ALL OCCURRENCES OF "@" CHARACTER TO "*".
* ---------------------------------------------------------------------
* ******************************************************** *
*   Licensed Materials - Property of IBM                   *
*                                                          *
*   5696-234                                               *
*                                                          *
*   (C) Copyright IBM Corp. 1975, 2004. All Rights Reserved. *
*                                                          *
*   US Government Users Restricted Rights - Use,           *
*   duplication or disclosure restricted by GSA ADP        *
*   Schedule Contract with IBM Corp.                       *
*                                                          *
* ******************************************************** *
 ON $ABEND GOTO LOGIT
 ON $CANCEL GOTO LOGIT
// UPSI 1
// OPTION JCANCEL,LOG,LINK,PARTDUMP
// LIBDEF *,SEARCH=(xref.test)                                          2
/*
// DLBL XRFLANG,'asmxref.langfile',,VSAM,CAT=cat_name,DISP=(OLD,KEEP)  3
// DLBL XRFTOKN,'asmxref.tokenfile',,VSAM,CAT=cat_name,DISP=(OLD,KEEP) 4
// DLBL XRFCF,'%%XRFCF',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=143
// DLBL XRFLOC,'%%XRFLOC',0,VSAM,DISP=(NEW,KEEP),                      C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=145
// DLBL XRFLOOC,'%%XRFLOOC',0,VSAM,DISP=(NEW,KEEP),                    C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=99
// DLBL XRFMDLO,'%%XRFMDLO',0,VSAM,DISP=(NEW,KEEP),                    C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192
// DLBL XRFSWU,'%%XRFSWU',0,VSAM,DISP=(,KEEP),                         C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=93
// DLBL XRFSWUO,'%%XRFSWUO',0,VSAM,DISP=(,KEEP),                       C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192
// DLBL XRFMWU,'%%XRFMWU',0,VSAM,DISP=(,KEEP),                         C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=96
// DLBL XRFTWU,'%%XRFTWU',0,VSAM,DISP=(NEW,KEEP),                      C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=80
// EXEC ASMXREF,SIZE=(ASMXREF,2M),PARM='options'                        5
*  SAMPLE CONTROL FILE FOR ASMXREF
*
  LIBRARY  LIB=xref.sample,TYPE=VSE,MEMTYPE=n                          6
    INCLUDE MOD=asmtest,LANGUAGE=language
*
*
  REPORT   REPORT=CF
  REPORT   REPORT=LOC
  REPORT   REPORT=LOOC
  REPORT   REPORT=MWU
  REPORT   REPORT=SWU
  REPORT   REPORT=TWU
  REPORT   REPORT=SOR
/*
IF $RC > 0 THEN
GOTO LOGIT
/*
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                      C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192
// DLBL SYSINDS,'%%XRFSWU',0,VSAM,DISP=(OLD,DELETE),                   C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=93
// EXEC ASMXREP,PARM='SWU 60 SYM'
/*
```

Figure 34 (Part 1 of 3). Sample ASMXREF VSE JCL

```
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192              |
// DLBL SYSINDS,'%%XRFMWU',0,VSAM,DISP=(OLD,DELETE),                     C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=96                |
// EXEC ASMXREP,PARM='MWU 60 MAC'                                         7
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192              |
// DLBL SYSINDS,'%%XRFCF',0,VSAM,DISP=(OLD,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=143               |
// EXEC ASMXREP,PARM='CF'                                                |
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192              |
// DLBL SYSINDS,'%%XRFLOC',0,VSAM,DISP=(OLD,KEEP),                       C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=145               |
// EXEC ASMXREP,PARM='LOC'                                               |
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
 * NOTE : IF "LOOC REPORT" IS REQUIRED. EXECUTE THE FOLLOWING            |
 *        STEP BY UNCOMMENTING THE FOLLOWING.                            |
 *        LOOC REPORT IS AVAILABLE FOR CPP PROGRAMS.                     |
 * // DLBL SYSINDS,'%%XRFLOOC',0,VSAM,DISP=(OLD,KEEP),                  C
 *               RECORDS=(1000,500),CAT=cat_name,RECSIZE=99              |
 * // EXEC ASMXREP,PARM='LOOC 60'                                        |
 * /*                                                                    |
 * IF $RC > 0 THEN                                                       |
 * GOTO LOGIT                                                            |
 /*                                                                      |
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192              |
// DLBL SYSINDS,'%%XRFTWU',0,VSAM,DISP=(OLD,KEEP),                       C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=80                |
// EXEC ASMXREP,PARM='TWU'                                               |
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT                                                               |
/*                                                                       |
// DLBL SYSPRT,'%%SYSPRT',0,VSAM,DISP=(NEW,KEEP),                        C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=8192              |
// DLBL IJSYSPH,'SOR.DATA',0,SD                                          |
// EXTENT SYSPCH,SYSWK2,1,0,5110,15                                       8
   ASSGN SYSPCH,DISK,VOL=SYSWK2,SHR                                      |
// DLBL SYSINDS,'%%XRFTWU',,VSAM,CAT=cat_name                            |
// EXEC ASMXREP,PARM='SOR'                                               |
/*                                                                       |
   CLOSE SYSPCH,FED                                                      |
/*                                                                       |
// DLBL IJSYSIN,'SOR.DATA',0,SD                                          |
// EXTENT SYSIPT,SYSWK2,1,0,5110,15                                       8
   ASSGN SYSIPT,DISK,VOL=SYSWK2,SHR                                      |
// EXEC LIBR,PARM='ACC S=XREF.XREFN;CAT SOR.A R=Y'                       |
/*                                                                       |
   CLOSE SYSIPT,FEC                                                      |
/*                                                                       |
IF $RC > 0 THEN                                                          |
GOTO LOGIT
```

Figure 34 (Part 2 of 3). Sample ASMXREF VSE JCL

```
/*
@ $$ PUN DISP=I,PRI=6,CLASS=A
// ASSGN SYSIPT,SYSRDR
// EXEC IESINSRT
#/ JOB ASMXRCAT
// EXEC LIBR,SIZE=256K,PARM='ACC S=XREF.XREFN'
@ $$ END
// UPSI 1
// DLBL XRFTWU,'%%XRFTWU',0,VSAM,DISP=(OLD,KEEP),CAT=cat_name
// EXEC DITTO,SIZE=512K
$$DITTO SC FILEIN=XRFTWU
/*
// EXEC IESINSRT
#&
@ $$ END
/*
/. LOGIT
// UPSI 1
// DLBL SYSINDS,'%%XRFTWU',0,VSAM,DISP=(OLD,DELETE),                    C
               RECORDS=(1000,500),CAT=cat_name,RECSIZE=80
// EXEC DITTO
$$DITTO SET DATAHDR=NO
$$DITTO SPR FILEIN=SYSINDS
/*
// EXEC LISTLOG
/&
@ $$ EOJ
```

Figure 34 (Part 3 of 3). Sample ASMXREF VSE JCL

Replace *cat_name*, throughout the JCL, with the name of your VSAM catalogs.

1 Replace *node* with the node, and *userid* with your user ID.

2 Replace *xref.test* with the name of the ASMXREF executable sublibrary.

3 Replace *asmxref.langfile* with the name of the XRFLANG file. See the note on page 120.

For details of the XRFLANG file see “ASMXREF XRFLANG Statements” on page 132.

4 You require this DLBL statement only for the TWU and SOR reports; you can remove it for other reports.

When you need to supplement, or replace, the default tokens with your own tokens replace *asmxref.tokenfile* with the name of the file containing your token statements. See the note on page 120. If the default tokens are sufficient, and additional tokens are not required, enter only a comment statement in the XRFTOKN file.

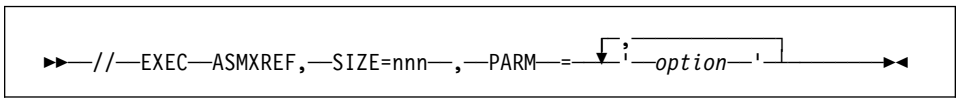
5 Replace *options* with any options that you need for your ASMXREF run.

The **EXEC ASMXREF** runs the program ASMXREF. ASMXREF needs you to specify at least one PARM option with the EXEC ASMXREF statement. The ASMXREF control statements can follow the EXEC statement in SYSIPT with each statement on a separate line, as shown in the sample JCL, or you can assign SYSIPT to a file containing the control statements. You must follow the last control statement with the SYSRDR termination control characters `/*`.

The sample JCL supplied with ASMXREF includes the statement MEMTYPE=A. If this is not correct change to the correct member type.

For details of the options available with ASMXREF, see “ASMXREF Options” on page 132, for details of the control statements, see “ASMXREF Control Statements” on page 125.

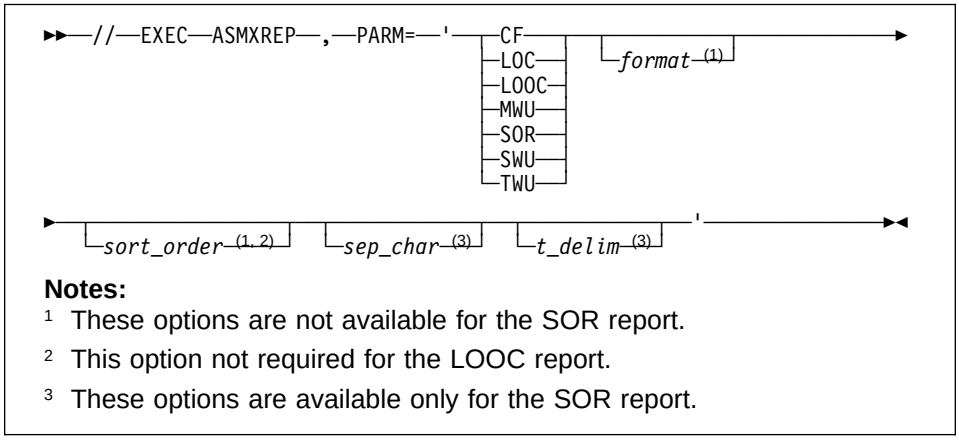
The format of the command is:



6 See “ASMXREF Control Statements” on page 125 for details of statements.

7 Enter the options required for each ASMXREP run. The sample JCL shows valid ASMXREP options.

You must run the **EXEC ASMXREP** for every report that you need. The format of the statement is:



Notes:

1 These options are not available for the SOR report.

2 This option not required for the LOOC report.

3 These options are available only for the SOR report.

Note: All the parameters are positional. You must enter them in the order shown above, or you can enter just the report parameter and leave the other parameters blank. If you enter the parameters in the wrong order ASMXREF issues an error message.

For details of the options available see “ASMXREP Options” on page 135, and for details of the reports available see “Understanding the Reports” on page 135. You must specify only one report with this statement.

8 Change the EXTENT cards to point to free space.

ASMXREF Control Statements

*

An asterisk (*) character in column one indicates a comment statement.

Library

You can abbreviate LIBRARY to **L**.

You must specify one, and only one, LIBRARY statement in the control file for each ASMXREF scan run. The minimum requirement with the LIBRARY statement is the LIB parameter that specifies the library ASMXREF scans. The following parameters are valid with the LIBRARY control statement:

LANGUAGE

Specifies the language of the input source files. You can specify only one language for each ASMXREF run, therefore all source files in the ASMXREF run must be in the same language. If you do not specify the LANG parameter ASMXREF uses the default language of assembler. The following table lists the supported languages and the associated keyword.

Table 18 (Page 1 of 2). XRFLANG Supported Languages

Language	**LANGUAGE=***keyword*
Assembler	ASM
Assembler 86	ASM86
C	C
C++	CPP
CLIST	CLIST
COBOL	COBOL
FORTRAN	FORTRAN
Generic	GENERIC
ISPF Panels	IPN
ISPF Panels	PANELS
ISPF Skeletons	SKE
ISPF Skeletons	SKELS
MASM	MASM
MODULA	MODULA
MODULA2	MODULA
MODULA3	MODULA
MVS or VSE JCL	JCL
OS/2 Command	CMD
OS/2 DEF	DEF
OS/2 IPF	IPF
OS/2 MAK	MAKE
OS/2 RC	RC

Table 18 (Page 2 of 2). XRFLANG Supported Languages

Language	LANGUAGE=*keyword*
OS/2 UID	UID
Panels	PANELS
PASCAL	PASCAL
PL/I	PLI
QMF™	QMFQUERY
QMFQUERY	QMFQUERY
REXX	REXX
RPG	RPG
SCRIPT	SCRIPT
SQL	QMFQUERY

LIB On MVS and VSE, this specifies the names of the PDS (MVS) or library (VSE) to be scanned. ASMXREF scans all source files in the PDS or library unless it encounters an INCLUDE or EXCLUDE control statement.

On CMS this specifies the name of the source list file that contains a list of the files to be scanned. The source list file must have a filetype of EXEC. For details on the source list file see “ASMXREF Source List File” on page 117.

The LIB parameter is required.

MEMTYPE
VSE only. Specifies the source member type. If you do not specify a source member type ASMXREF defaults the member type to A for the library specified with the LIB parameter.

TYPE The type of library ASMXREF scans. Valid types are:

- **PDS** (Default) An MVS partitioned data set.
- **SEQ** An MVS sequential data set.
- **VSE** A VSE Librarian library. ASMXREF requires this when running on VSE.
- **CMS** A CMS file. ASMXREF requires this when running on CMS.

MVS Example:

```
LIBRARY LANGUAGE=COBOL,LIB=COBOL.SOURCE,TYPE=PDS
```

VSE Example:

```
LIBRARY LANGUAGE=ASM,LIB=COMMON.XREF,TYPE=VSE,MEMTYPE=D
```

CMS Example:

```
LIBRARY LANGUAGE=PLI,LIB=SOURCE,TYPE=CMS
```

Include

You can abbreviate INCLUDE to **I**.

On MVS and VSE the INCLUDE statement defines the members of the PDS (MVS) or library (VSE), specified with the LIBRARY LIB statement, that ASMXREF includes in the scan. When you specify a member with the INCLUDE statement

ASMXREF scans only that member. If the INCLUDE statement is omitted, ASMXREF scans all the members in the specified library except those members excluded with the EXCLUDE statement.

ASMXREF ignores the INCLUDE statement on CMS as ASMXREF scans only source files listed in the source list file.

Specify the following parameter with the INCLUDE control statement:

MOD Specifies the module to include.

Example:

```
INCLUDE MOD=FILENAME
```

Exclude

You can abbreviate EXCLUDE to **E**.

On MVS and VSE the EXCLUDE statement defines the members of the PDS (MVS) or library (VSE), specified with the LIBRARY LIB statement, that are excluded from the scan. If the EXCLUDE statement is omitted, ASMXREF scans all the members in the PDS or library specified with the LIBRARY LIB statement, unless you specify an INCLUDE statement.

ASMXREF ignores this statement on CMS as ASMXREF scans only source files listed in the source list file.

Specify the following parameter with the EXCLUDE control statement:

MOD Specifies the module to exclude.

Example:

```
EXCLUDE MOD=FILENAME
```

Parm

You can abbreviate PARM to **P**.

Overrides processing default values. The following parameters are valid with the PARM statement:

ITBSIZE Default 50 000. Maximum number of tokens that ASMXREF can handle for one source statement. Minimum number 500.

LOGSIZE Default 50 000. Maximum number of characters that ASMXREF can handle in one statement. Minimum number 1 000.

MWUSIZE Default 1 000. Maximum number of macros that ASMXREF can handle in the Macro Where Used (MWU) report. Minimum number 100.

| **OOSIZE** Default 2 000. Maximum number of OO objects and classes that that ASMXREF can handle for each module processed. Minimum number 100.

SWUSIZE Default 10 000. Maximum number of symbols that ASMXREF can handle in the Symbol Where Used (SWU) report. Minimum number 500.

Example:

```
PARM ITBSIZE=100000
```

Report

You can abbreviate REPORT to **R**.

Specifies the format of the reports you require. ASMXREF requires at least one REPORT statement in the control file. Specify the following parameter with this control statement:

REPORT The name of the required report. Valid reports are:

CF	Control Flow Report
LOC	Lines of Control Report
LOOC	Lines of OO Code Report
MWU	Macro Where Used Report
SOR	Spreadsheet Oriented Report
SWU	Symbol Where Used Report
TWU	Token Where Used Report

You can specify only one report on each REPORT statement. For details on the reports available see “Understanding the Reports” on page 135. To specify more than one report create a REPORT control statement for each desired report.

Example:

```
REPORT REPORT=TWU
REPORT REPORT=MWU
```

ASMXREF Token Statement

A token is an arbitrary string of characters specified for inclusion or exclusion in the ASMXREF scan. The TWU, SOR and TSP reports show occurrences of all include tokens, unless ASMXREF matches a retrieved token with an exclude token. ASMXREF does not use tokens in other reports.

If the language is case-insensitive, ASMXREF converts source records to uppercase internally. This simplifies the matching process.

The XRFLANG file supplied with ASMXREF lists the default tokens included in a scan. When you create the TWU or SOR reports ASMXREF processes the default tokens unless you enter a TOKEN NODEFLT statement in the XRFTOKN file. For details on the TOKEN NODEFLT statement see page 130. The default tokens have been designed primarily for the assembler language and represent most fields of interest, in that language. You can modify the default token list to include tokens more suited to your site. For details of the default tokens supplied with ASMXREF, and modifying those tokens, see “ASMXREF XRFLANG Statements” on page 132.

The default tokens may be sufficient for all your ASMXREF scans, but you may need to supplement, or replace, the default tokens in the XRFLANG file with your own tokens for a scan with special requirements. When you need to do this it is not advisable to modify the default token list, but instead create your own XRFTOKN file.

You must enclose all tokens in the XRFTOKN file within matching delimiter characters. The delimiter character can be any non-space character but the start and end delimiter must be the same. You can use a different pair of delimiter characters for each token. For example, if you have a double quote (") character

embedded in the token, such as MM"DD, then you can use the ? character as the delimiter and then use the " for the next token. For example:

```
?MM"DD?
"YY/MM"
```

For details of the scanning rules ASMXREF applies see “Scanning Rules for ASMXREF” on page 131.

Token

You can abbreviate TOKEN to **T**.

The following parameters are valid with the TOKEN statement:

INCLUDE You can abbreviate INCLUDE to **INC**.

Specifies a token to include in the scan for the TWU, SOR and TSP reports.

To specify a token explicitly enter the exact search token between matching delimiter characters. For example:

```
TOKEN INC="DATE/TIME"
```

ASMXREF scans all the source files specified with the ASMXREF control statements, searching each for an exact match with the specified explicit tokens.

You can also specify the INCLUDE token generically with a mask character inserted in the search token. ASMXREF treats the mask character as a wildcard and retrieves all, or any, characters in the position of the mask character. The default mask character is the asterisk (*) that represents any number of characters (including none). For example:

```
TOKEN INC="DATE/*I*"
```

retrieves:

```
DATE/TIME
DATE/LINE
```

ASMXREF allows spaces within the token string but does not accept them between the parameters and the start of the token string. The following example is acceptable:

```
TOKEN INC='ab c'
```

The following example is not acceptable:

```
TOKEN INC= 'ABC'
```

EXCLUDE You can abbreviate EXCLUDE to **EXC**.

Specifies a token to exclude from the scan for the TWU, SOR and TSP reports.

When a TOKEN INCLUDE statement contains a generic mask (wildcard) character, the TOKEN EXCLUDE statement specifies the exclusion of the token when it is found by the INCLUDE statement token. You cannot enter a generic mask character in a TOKEN EXCLUDE statement.

Note: The TOKEN EXCLUDE statement only applies to the previous TOKEN INCLUDE statement that must contain a generic mask. If you need to repeat the TOKEN EXCLUDE statement, for another TOKEN INCLUDE statement, then you must repeat the token exclude statement.

Example:

```
TOKEN INC="DDMM*"
TOKEN EXC="DDMMCCYY"
```

retrieves:

```
DDMMYY
DDMMM
```

ASMXREF does not report the following string because it matches the exclude token:

```
DDMMCCYY
```

MASK Specifies a wildcard character.

The asterisk (*) character is the default generic mask (wildcard) symbol.

If you enter a search token that contains the mask character itself you must specify an override to the mask character, with the MASK parameter. This is applicable only to the previous TOKEN INCLUDE statement.

If multiple MASK parameters are entered together, ASMXREF uses only the last one for the previous INCLUDE token. The following example shows the MASK parameter:

```
TOKEN INC="DA%E/*IM%"
TOKEN MASK="%"
```

retrieves:

```
DATE/*IMAGE
DANE/*IMAGINARY
```

In the previous example ASMXREF takes the % character as the mask symbol for the previous TOKEN INC statement.

If you enter this statement:

```
TOKEN INC="***"
```

ASMXREF treats the statement as an explicit token and retrieve all occurrences of ***.

Note: The TOKEN MASK statement only overrides the default for the previous token statement. On finding another token statement ASMXREF reapplies the default value of a * representing the mask character.

NODEFLT Turns off processing of the default tokens supplied in the XRFLANG file. This statement does not affect the processing of the language-specific exclusion verbs.

Note: To create the TWU and SOR reports ASMXREF must have tokens specified. If you turn off processing of the default tokens, with the TOKEN NODEFLT statement, you must supply tokens in the

XRFTOKN file. If you do not specify a XRFTOKN file ensure default tokens exist in the XRFLANG file.

Example:

```
TOKEN NODEFLT
```

NOSEP Suppresses the separator records in the Tagged Source Program (TSP).

Example:

```
TOKEN NOSEP
```

ASMXREF creates separators by default and saves them in the Tagged Source Program (TSP) that it creates in the scan phase. Producing separators allows this file to be split into individual members that you can use to replace or create macro or copy libraries. For details on splitting the TSP see “Tagged Source Program (TSP)” on page 155.

Scanning Rules for ASMXREF

The token control statements define the tokens included in or excluded from the scan. This section explains the rules applied by ASMXREF with the token statement.

Generic Matching Rules

You can specify a token with the mask character in the first character or last character position. ASMXREF then searches for a match on any number of characters before or after the token specified. If ASMXREF finds a match, in the source record for the token, it scans forwards and backwards from the match to the scan end character. The scan end character is a space, `' '`. ASMXREF passes the space delimited match to the matching process of the EXCLUDE tokens. If ASMXREF matches the retrieved token with an exclude token it excludes the match from the report.

Here is an example to help you understand this rule:

Source Record

```
MVC DATE(8),SYSDATETIME
```

ASMXREF control statements

```
TOKEN INC="*DATE*"
TOKEN EXC="DATETIME"
```

After finding the first match ASMXREF restarts the scan from the character following the token. In the previous example the ASMXREF scanning process finds the first occurrence of `DATE`. As the token is specified generically with an `*` in the first and last character ASMXREF scans forwards and backwards from the match, until it encounters the space scan end characters. ASMXREF retrieves the string `DATE(8),SYSDATETIME`. ASMXREF then continues the scan from the character following the first `DATE` match, which in the example is the `'('`, until it finds the second match. Again ASMXREF scans forwards and backwards from this match, until it encounters the space characters. The second match again retrieves the string `DATE(8),SYSDATETIME`. ASMXREF compares the retrieved string with the exclude statement, which in this example does not apply to either match.

If you had specified the following exclude statement:

```
TOKEN EXC="DATE(8),SYSDATETIME"
```

ASMXREF excludes the two matches from the TWU, SOR or the TSP reports.

Another example:

Source Record

```
GETMAIN R,LV=(0),LOC=BELOW
```

ASMXREF control statements

```
TOKEN INC="*LV=(0)*"
TOKEN EXC="R,LV=(0),LOC=BELOW"
```

When ASMXREF finds the match it scans forwards and backwards until it encounters the space end characters. In the previous example the match is `"R,LV=(0),LOC=BELOW"`. The exclude token `"R,LV=(0),LOC=BELOW"` matches the retrieved token and so ASMXREF excludes the match.

ASMXREF Options

On MVS and VSE, you specify ASMXREF options with the PARM parameter. On CMS, you specify ASMXREF options in a file with the same filename as the control file, and a filetype of DEFAULTS.

You must specify at least one option:

DUP (CMS Only) Process modules with duplicate names.

NODUP Do not process source files with duplicate names. This is the default.

MSGLEVEL The lowest level of messages ASMXREF is printed. Range is 0 to 16. The default is 0.

PAGELEN Specifies the page length for the SYSPRINT file. The default is 55.

ASMXREF XRFLANG Statements

ASMXREF is supplied with a sample language (XRFLANG) file. This file contains two segments:

1. The default token segment contains the tokens included by default in the ASMXREF scan phase for the TWU and SOR reports.
2. The language segment contains the languages supported by ASMXREF, and the language-specific exclude verbs (words). ASMXREF treats these verbs in a similar way to exclude tokens, but excludes them when scanning a file in the specified language.

```
DEFAULT TOKENS
DATE
TIME
MM/DD/YY
MM/YY
DD/MM/YY
YY/MM/DD
YYDDD
MONTH
DAY
YEAR
YR
*DATE
DAT*
*YR'
*C''20*
*P''20*
LANG=FORTRAN
ARRAY
BACKSPACE
⋮
LANG=ASM
DC
EQU
⋮
LANG=COBOL
ACCEPT
ACCESS
⋮
LANG=C
&&
_ANSI_
⋮
LANG=CPP
&&
_cplusplus
LANG=PLI
%ACTIVATION
%DECLARE
⋮
LANG=RPG
*IN0A
*IN0B
⋮
```

Figure 35. Sample XRFLANG File

Default Token Segment

The default token segment contains a header in the format:

```
DEFAULT TOKENS
```

A list of the default tokens follows the header record. The list contains one token on each line, each starting in column one. The sample XRFLANG file supplied with ASMXREF contains a set of default tokens. These tokens are designed for the

assembler language but you can modify them to suit your environment. You can enter either explicit tokens (exactly as you need the token) or generic tokens where you use the asterisk (*) character as a wildcard. ASMXREF does not treat a mask character, in the XRFLANG file, as a space character, as it does when you use a mask character in the XRFTOKN file.

When adding default tokens you do not need to enter the TOKEN INCLUDE statement and you do not need to enclose the token in delimiters.

ASMXREF treats all tokens in the default token list as include tokens and does not accept exclude tokens in the XRFLANG file.

If you need a token list for your own run, it is better to create your own XRFTOKN token statement file containing your personalized tokens, rather than modifying the default list. For details on how to create an XRFTOKN file see “ASMXREF Token Statement” on page 128.

If you need to turn off processing of the default tokens create your own XRFTOKN file and enter a TOKEN NODEFLT statement in the file. Remember the TWU and SOR reports require tokens, therefore, you must have tokens in either the XRFTOKN file or the XRFLANG file.

For a list of the default tokens see Figure 35 on page 133.

Note: These tokens are designed for the assembler language. If you are using another language, modify them to suit your environment.

Language Segment

The language segment contains a header in the format:

```
LANG=nnnnnnnn
```

where *nnnnnnnn* is the keyword representing a language supported by ASMXREF. ASMXREF supports all the languages listed in this file; you cannot add other languages to this file.

When you run the ASMXREF scan phase you must specify the language of the files in the library ASMXREF is scanning. Do this with the `LIBRARY LANGUAGE=`*`language_name`* control statement. The *language_name* is the keyword of the language, as specified in the XRFLANG file. For details on the LIBRARY control statement see “ASMXREF Control Statements” on page 125.

A list of exclude verbs (words), applicable to the language, follow the language header. The ASMXREF scan excludes these verbs when it creates the TWU or SOR reports, when the source files are in the language specified in the header record. The sample XRFLANG file supplied with ASMXREF contains a sample set of language-specific exclude verbs. After ASMXREF is installed, update this file and modify the exclude verbs to suit each language used in your environment. Specify the exclude verbs one per line starting in column one.

ASMXREP Options

This section describes the options available with ASMXREP. On MVS you specify ASMXREP options with the RPARM parameter; on VSE you specify ASMXREP options with the PARM parameter; on CMS you specify ASMXREP options in a file with the same filename as the control file and a filetype of DEFAULTS.

Format *nnn* (Default 60). Printer format, where *nnn* is the number of lines per page. Not available with the SOR report. Enter any number between 20 and 999. You do not need to enter a leading zero. You can not use this option with the SOR report.

Sort Order The order in which ASMXREF sorts the report. You cannot use this option with the SOR report.

MAC Generates the CF report in macro order.
PART Generates the CF report in module order.
MOD Generates the LOC report in module order.
COM Generates the LOC report in component order.
MAC Generates the MWU report in macro order.
MOD Generates the SWU report in module order.
PART Generates the MWU report in module order.
SYM Generates the SWU report in symbol order.
SYMC Generates the SWU report in compact symbol order.

You can not use this option with the SOR or TWU reports.

SEP_CHAR You can only use this option with the SOR report. The separator character for building the spreadsheet cells. The separator can be only a comma or semicolon. Default is **,** (comma).

T_DELIM You can only use this option with the SOR report. The title delimiter for each token cell. The title delimiter can be any single character recognized by the spreadsheet application. Default is '(apostrophe).

Understanding the Reports

This section describes the reports available in ASMXREF and provides the following information:

- The languages supported by each report
- A description of each report
- A sample of each report

Languages Supported by Reports

Table 19. Languages Supported by ASMXREF Reports

Language	CF	LOC	LOOC	MWU	SOR	SWU	TWU
ASM370	√	√		√	√	√	√
ASM86	√	√		√		√	
C	√	√		√	√	√	√
C++	√	√	√	√	√	√	√
CLIST		√					
COBOL		√			√		√
FORTRAN		√			√		√
Generic		√					
ISPF		√					
JCL		√					
MASM		√					
MODULA 2/3		√					
OS/2 cmd		√					
OS/2 DEF		√					
OS/2 IPF		√					
OS/2 MAK		√					
OS/2 RC		√					
OS/2 UID		√					
PASCAL		√					
PL/I	√	√		√	√	√	√
QMF/SQL		√					
REXX	√	√		√	√	√	√
RPG		√			√		√
SCRIPT		√					

Control Flow (CF) Report

The Control Flow report tabulates all intermodule program references as a function of member or entry point name. It can list references either in the order of the members *referring to* the subject entry point or the entry point names *referred by* the subject member, depending on the sort order.

For each part processed, the CF report can handle up to 256 internal procedure names and 1024 entry point names.

Reference names that exceed 64 characters are truncated.

ASMXREF classifies each references by type. The classification is language specific and is described in the following sections.

C Family References

ASMXREF scans the following C statements to extract Control Flow information:

- Function declarations
- Function definitions
- Macro definitions
- Expressions in the each statement.

Note: A sample C program is shown in Figure 36 on page 138, with accompanying CF report in Figure 37 on page 139.

Functional references for C code are classified as follows:

- Defined macros are identified and flagged as #define type references.
- Declared functions are recognized as declarative.
- Extern functions that are called in different expressions are identified as CALL type of references.
- Static functions and #defined macros that are called from different expressions are recognized as Local Calls.
- Functions defined as extern functions within the module are identified as such.
- Functions defined as static functions within the module are flagged as static definitions.

Notes:

1. For #define and function definitions, the references are assumed to be made from the module.
2. For Call and Local Call type of references the references are assumed to be made from calling functions.
3. A statement of the following format is always treated as a function call unless *symbol1* is a generic data type (char, int, etc.) in C.

```
symbol1 ( *symbol2 ) ;
```

PL Family References

ASMXREF determines references and their types by analyzing the following PL instructions:

- CALL
- ?LINK
- ?LOAD
- ?XCTL
- ?ATTACH
- DCL .. NONLOCAL EXTERNAL (EXTERN)
- DCL .. LOCAL EXTERNAL (EXTERN)
- EXIT TO (VCON)

External declarations are extracted as EXTERN type references if they are qualified by LOCAL or NONLOCAL attributes. The labels in the EXIT TO instructions are identified as VCON references. All other reference types are classified per the instruction names.

REXX References

For REXX programs, ASMXREF analyzes the following REXX instructions to extract Control Flow references:

- Call
- Signal
- Function Invocation

All reference types are CALL, and functions are assumed to be external unless the function name is found in a Procedure statement, in which case it is flagged as LOCAL.

```
/* Physical file name : moda.c                    */
#define max(a,b)  (a>b ? a : b)

int FunctionA(int a )
{
  a = ProcessA(a) ;
  if (ProcessB())
     return(0) ;
  while( ProcessC() )
     printf("please wait \n") ;
  a = max(a,0);
}

static int ProcessA(int a)
{
     return(a) ;
}

extern int ProcessB()
{
     return(0) ;
}

int ProcessC()
{
     return(0) ;
}
/* end of module MODA        */
```

Figure 36. Sample C Program Used for CF Report

```
Date: 07/09/2004      ASMXREF V1.5.0 Control Flow Report                    Page      1
Time: 11:50:36     Sorted by Referenced Function/Module (MAC)
                       Includes ALL Symbols (EXT and INT)

Referenced Function/Module
    Ref Keys   Calling Functions/Modules                                        #

max
     C         FunctionA                                                      1
    V          MODA C                                                         1

printf
     C         FunctionA                                                      1

FunctionA
          E    MODA C                                                         1

ProcessA
     C     L   FunctionA                                                      1
      D    L   MODA C                                                         1

ProcessB
     C         FunctionA                                                      1
          E    MODA C                                                         1

ProcessC
     C         FunctionA                                                      1
          E    MODA C                                                         1
-------------------------------------------------------------------------------------------
C-Lang: I=Invalid C=Call D=Static Def K=Dcl  L=Static V=#def E=Extrn Def
Others: A=Attach  C=Call D=Load       K=Link L=Local  V=VCON E=Extrn     X=XCTL
```

Figure 37. Sample CF Report

Lines Of Code (LOC) Report

The Lines Of Code report provides a count, arranged by part and by component, of:

- Number of source lines in the part.
- Number of comments in the part.
- Shipped source instructions (SSI), which are the number of instructions within each part scanned, both executable and non-executable, that are not spaces or comments.
- Changed source instructions (CSI), which are the number of unique SSI that have been modified in each part categorized by added, changed, deleted, moved, etc.

In addition, the LOC Report provides a summary report of CSI arranged by programmer.

Note: CSI counts are provided only for changes that are *marked* using the standard flags as described in “Changed Source Instruction (CSI) Measurements” on page 140.

A sample LOC report is shown in Figure 38 on page 140. In this sample, the Release 5.0 and **Origin/Programmer** flags were allowed to default to `ALL`.

```
Date: 07/09/2004          ASMXREF V1.5.0 Lines of Code Report     Page      1
Time: 11:55:06                          by Module

Product    = SAMPLE
Release    = ALL
Programmer = ALL

                                                                (ADD+CHG)
     Module            Language Records Comments      SSI      CSI    ADDED  CHANGED  DELETED    MOVED   COPIED

   1 XREFTST1 PLX      PLX           54       26       24       23       21        2        5        1
   2 XREFTST2 ASSEMBLE ASM           58       17       41       40       35        5        5        1
   3 XREFTST3 CXX      CPP           84       43       40       40       34        6       14

PRODUCT TOTALS:                     196       86      105      103       90       13       24        2


REPORT TOTALS:                      196       86      105      103       90       13       24        2




Date: 07/09/2004          ASMXREF V1.5.0 Lines of Code Report     Page      2
Time: 11:55:06                Programmer Summary Report

Release    = ALL
Programmer = ALL

                                                                (ADD+CHG)
        Programmer                                                 CSI    ADDED  CHANGED  DELETED    MOVED   COPIED

        ANYCODER  .....................................          3        1        2                1
        DEPT01    .....................................         33       33
        DEPT28    .....................................          3        1        2                1
        GER       .....................................          1                 1       19
        RAS       .....................................          4        1        3        5
        ROBINS    .....................................         59       54        5

PROGRAMMER TOTALS:                                            103       90       13       24        2
```

Figure 38. Sample LOC Report

Changed Source Instruction (CSI) Measurements

This section describes the coding standards required to obtain CSI measurements.

Comments, Unit Descriptors, Change-Flag Descriptors, and Change Flags:
This section describes the volume measurement rules for:

- Comments
- Unit Descriptors
- Change-Flag Descriptors
- Change Flags.

These rules apply to all languages.

Comment Definition: Comments are categorized as block or remark.

1. Full-line comments

   A line that contains only commentary is a full-line comment. A full-line comment that is **not** imbedded within an instruction is counted as a comment. A block comment may span several lines.

   For nested comments, the entire text of the comment is regarded as block or remark based on whether the outermost comment starts as the first item on a line or not. The comment delimiters for inner comments are disregarded.

2. Remarks

A *remark* is any comment which is not a block comment. This is valid regardless of whether the comment appears within an instruction or at instruction boundary. A remark may also span several lines.

ASMXREF does **not** count remarks.

Note: Blank lines are counted as comments.

Unit Descriptor: A unit descriptor gives the name of the module, and the names of the component and product containing the module. Unit descriptors are not required in your code, but are recommended to provide Component and Product classifications for the LOC report.

A unit descriptor has one of the following formats depending on whether the source is a module, segment, or macro:

```
$MOD(unitname) COMP(component) PROD(product) :  comment
$SEG(unitname) COMP(component) PROD(product) :  comment
$MAC(unitname) COMP(component) PROD(product) :  comment
```

If present, the unit descriptor must be the first item on any line of a block comment. Furthermore, there should not be more than one unit descriptor in a source file. If more than one is found, only the first one is used. A unit descriptor cannot be split over several lines.

Unit, component, and product names are enclosed in parentheses and can consist of any character other than the closing parenthesis. The maximum size of each of these names is 8 characters.

The keywords may be separated by either a space or a comma.

The following example describes a unit descriptor for the unit ADDPROC, which is a module. It belongs to the component SC123, which is part of the product XYZ:

```
$MOD(ADDPROC) COMP(SC123) PROD(XYZ): Add Procs to Test File
```

Figure 39. Sample Unit Descriptor

Change-Flag Descriptor:

Change-flag descriptors are used to group all changes made for a particular reason qualified by the release number, date, and origin associated with those changes.

Change-flag descriptors are also used to define implicit change flags which indicate the number of SSIs that have not been changed.

The format of a standard change-flag descriptor is:

```
$pn= reason release# date origin : comments
```

where `p` is the process class and `n` is the index of the flag. The process class can be used to determine a specific type of change activity.

The following table lists recommended conventions for process class codes:

Note: Process class codes are not limited to these, and each location or development team may chose to create their own scheme for categorizing

changes. Many groups just start with any alphanumeric flag (for example A1, AA, 11) and increment as needed, with the exception of an index of zero, which is reserved for implicit flags.

Table 20. Process class code conventions

Class code	Class name
D,E,F,G	DCR Design Change
H,I,J,K	HDWE Hardware Support Change
L,M,N,O	LINE Line Item
P,Q,R,S	PTM Internal Problem Reports
0-9	APAR User Problem Reports

The reason, release#, date, and origin fields may be separated by spaces or commas.

The change-flag descriptor must be the first non-space item on any line of a block comment. A block comment can contain more than one change-flag descriptor, each appearing on a different line.

The following table describes each field of a change-flag descriptor:

Table 21. Definition of the Change-Flag-Descriptor Fields

Field	Length	Usage
$	1	A delimiter dollar character.
pn=	3	For the flag ID, where p is the process class and n is the index for the specified process class.
reason	1 to 12	The reason for the change, for example, the number of a line item, APAR number, or PTM/PTR number.
release#	1 to 8	A release number, for example, `041` for Release 4.1.
date	0 to 8	A date in any desired format. For example, `930901` in Gregorian format, or `09/01/93` in US format.
origin	0 to 8	Information about the origin of the set of changes, for example, the initials, name, or user ID of the requester.
:	1	A delimiter colon character.
comment	0 to 80	Any explanatory text.

The reason, release, date, and origin fields can consist of any sequence of characters except a space, comma, or colon. If the length of a field exceeds the permissible range, the field is truncated. However, if the length of a field is less than the maximum, it is padded with spaces on the right.

Date and origin are optional fields, but if a particular field is specified, all the fields to its left must also be present.

Note: ASMXREF searches for flag descriptors throughout the module. If the ending delimiter ":" is missing, ASMXREF recognizes the descriptor but issues an error message.

Flag Descriptor for Implicit Flagging: If a standard change-flag descriptor defines a process code with an index value of zero, the descriptor defines an implicit change flag.

Implicit flagging refers to the automatic application of a change flag to all the SSIs that are not changed (that is, to all unflagged instructions). For example, if the flag descriptor defines the process code as h0 as in the following example, then ASMXREF assumes that all unflagged instructions in that module are flagged with the implicit flag H0.

A unit should contain only one change-flag descriptor that defines an implicit change flag. If a unit contains more than one implicit definition, only the first one is accepted and the rest are ignored.

The following figure gives some examples of valid change-flag descriptors:

```
FLAG REASON    RLSE   DATE     ORIGIN   COMMENTS
---- --------  ----   -------- -------  --------------------
$H0= DA24     ,033   ,760718,  RM44   : CREATED
$02= ZA34537   811,   770416   DDR    : CORRECT BALANCE
$P2= PTR0336   983    871211          : FIX BLOCK ESCAPE
$h1= SKI1223A  103    09/01/93 Sharon : Increase Date Field
```

Figure 40. Sample Change-Flag Descriptors

Change Flags: Change flags are used to mark all changes in a source file made during development and maintenance.

Standard change flags have the following format:

```
m@pnc
```

where

```
m     is an optional multiplication factor
@     is the @ sign itself
p     is the process class
n     is an index
c     is the change code
```

The process class, index, and change code may be alphanumeric.

The multiplication factor must be numeric and can be used only for delete flags.

Note: An exception to this rule is change flagging for languages that do not permit remarks, like ISPF and COBOL. Consequently, summary flags may be used to describe the number of instructions changed. For these programs, a multiplication factor may be specified for any change code. The multiplication factor, however describes the number of instructions changed and not the number of source lines changed.

The change code can be any of the following:

A Add

C Change

P Copy

M Move

D Delete

Change flags for deleted instructions are usually coded within a block comment. The multiplication factor specifies the number of instructions that have been deleted. For example, “19@H1D” implies that nineteen instructions were deleted for the change defined by the flag descriptor H1.

Note: The delete count does **not** contribute to CSI.

No imbedded spaces are allowed within any change flag.

Change flags are identified as a sequence of characters starting with `@` and followed by three characters.

As mentioned above, an optional multiplication factor may be specified with a delete change flag only. If the multiplication factor is found on a non-delete flag in a language that supports remarks, it is ignored.

A change flag can be coded in a block comment or in a remark. Several change flags can be coded in a comment. The change flags must be the last non-space sequence of characters within a comment regardless of whether the comment spans several lines or not.

Rules for Counting Change Flags:

1. Only one change flag in a comment qualifies for the CSI count. For a comment containing several change flags, only the last (rightmost) change flag is counted.
2. A change flag of non-delete type is associated with the number of preceding language instructions ending on the line on which the comment started. If the number of instructions ending on that line is zero, the CSI count is zero. If more than one instruction ends on the same line, the flag is associated with all these instructions.

   Note: The LOC report does not recognize non-standard change flags.
3. For a standard change flag of delete type (change code D), a multiplication factor of one is assumed if not specified.

The following figure shows a sample XREF source header:

```
/*--------------------------------------------------------------------*/
/*                                                                    */
/* XREF Information :                                                 */
/*                                                                    */
/* $MOD(XREF) COMP(XREF) PROD(XREF): Main Logic for Scan Phase        */
/*                                                                    */
/* Flag Reason   Rlse Date   Origin  Flag Description                 */
/* ---- -------- ---- ------ ------  ------------------------------ */
/* $L0= F00000   070  770101 PAS   : Base Code                       */
/* $D1= F00093   093  851131 RLS   : Release 9.3                     */
/* $D2= LX0011   099  910304 Robins: Release 9.9                     */
/* $H1= LX0222   103  930630 Chip  : Support OS/2 Platform Version   */
/* ------------------------------------------------------------------ */
/*                                                                    */
/* Flag Format is @XNT Where:                                         */
/*                                                                    */
/* Suggested Indicators for X, but not required:                      */
/*                                                                    */
/* X = D,E,F,G  for DCR  - Design Change                              */
/*     H,I,J,K  for HDWE - Hardware Support Change                    */
/*     L,M,N,O  for LINE - Line Item                                  */
/*     P,Q,R,S  for PTM  - Internal Problem Reports                   */
/*     0-9      for APAR - User Problem Reports                       */
/*                                                                    */
/* N = any number or letter (ONLY use a '0' once per file, used       */
/*                           on all unflagged lines by XREF)          */
/* T = A - for Added code.                                            */
/*     C - for Changed code.                                          */
/*     D - for Deleted code (###@XNT) ### = number of lines deleted   */
/*     M - for Moved code.                                            */
/*     P - for Copied code.                                           */
/*                                                                    */
/*--------------------------------------------------------------------*/
```

Figure 41. Sample XREF Header

The LOOC report

The Lines of OO Code (LOOC) report gives the following information about C++ classes and objects:

- Lines of Code (LOC) per Class
- Lines of Code (LOC) per Object
- Objects per Class

All three sections are contained in the LOOC report file. The information used to build these three sections of the LOOC report is gathered from the LOOC and SWUO intermediate files.

Note: If the SWUO intermediate file is empty or missing, only the LOC per Class section is generated, and a message appears in the report to this effect. To obtain complete LOOC detail, you must request the SWU report during the Scan Phase, in addition to the LOOC report.

The LOC per Class section

The LOC per Class section contains the following information, arranged by class name:

- Number of comment records in the class declaration
- Number of SSI executable statements in the class declaration
- Number of SSI non-executable statements in the class declaration
- Module name in which the class declaration resides

The counts for each class include LOC for the class declaration itself and LOC for all member functions, whether or not the member function is declared within the class declaration.

Note: When a member function of a class is declared in a different module from the one in which the class is declared, the counts for the class and the member function remain separated.

A sample LOC per Class section is shown in Figure 42.

```
Date: 07/09/2004          ASMXREF V1.5.0 LOC Per Class Report     Page      1
Time: 12:00:35

Class
   Comments     SSI_X     SSI_N  Module

aBase
          0         4         0  CADDADD CPP
          0         4         0  CADDADD2 CPP
aClass
          0        14         0  CADDADD CPP
          0        14         0  CADDADD2 CPP
```

Figure 42. Sample LOC per Class section

The LOC per Object section

The LOC per Object section contains the following information, arranged by object name:

- Number of comment records
- Number of executable SSI
- Number of non-executable SSI
- Module name in which the object is declared, defined, and/or referenced.

The counts for each object are obtained by multiplying the number of times the object is referenced by the number of comments, SSI-executable, and SSI-nonexecutable records in the declaration of the class of which the object is an instance.

A sample LOC per Object section is shown in Figure 43 on page 147.

```
Date: 07/09/2004          ASMXREF V1.5.0 LOC Per Object Report    Page      2
 Time: 12:00:35

 Object
    Comments     SSI_X     SSI_N  Module

 Mine
           0         4         0  CADDADD CPP
           0         4         0  CADDADD2 CPP
 Mine2
           0         4         0  CADDADD2 CPP
 Yours
           0        14         0  CADDADD CPP
           0        14         0  CADDADD2 CPP
 Yours2
           0        14         0  CADDADD2 CPP
```

Figure 43. Sample LOC per Object section

The Objects per Class section

The Objects per Class section lists, for each class name, all of the objects which are instances of that class, and the modules in which the objects are referenced.

A sample LOC per Class section is shown in Figure 44.

```
 Date: 07/09/2004          ASMXREF V1.5.0 Object/Class Report      Page      3
 Time: 12:00:35

 Class
    Object                                                          Module


 aBase
    Mine                                                            CADDADD CPP
    Mine                                                            CADDADD2 CPP

 aClass
    Yours                                                           CADDADD CPP
    Yours                                                           CADDADD2 CPP
```

Figure 44. Sample Objects per Class section

Macro Where Used (MWU) Report

The Macro Where Used (MWU) report lists all macros invoked and all segments copied and included.

On CMS ASMXREF creates the MWU in a file named *filename* OUTMWU A, where *filename* is the name of the control file.

On MVS, the default name for the MWU, defined in the procedure supplied with ASMXREF, is *userid*.XREFOUT.MWU.

On VSE the MWU is printed from SYSPRT.

The report includes the type and frequency of the invocation/reference.

```
Date: 07/09/2004         ASMXREF  V1.5.0  Macro Where Used Report        Page      1
Time: 12:07:02                       Macro to Part Mapping - All Macros

Macro
   Module                  #  Type

FREEMAIN
   ASMTEST ASSEMBLE        1  ASM MACRO

   TOTAL                   1

GETMAIN
   ASMTEST ASSEMBLE        1  ASM MACRO

   TOTAL                   1

TIME
   ASMTEST ASSEMBLE        2  ASM MACRO

   TOTAL                   2

WTO
   ASMTEST ASSEMBLE        1  ASM MACRO

   TOTAL                   1
```

Figure 45. Sample Macro Where Used (MWU) Report

Spreadsheet Oriented Report (SOR)

The Spreadsheet Oriented (SOR) report is a comma-delimited file that you can import into a spreadsheet application, such as Lotus 1-2-3, to estimate effort and impact assessment.

On CMS ASMXREF creates the SOR in a file named *filename* OUTSOR A, where *filename* is the name of the control file.

On MVS, the default name for the SOR, defined in the procedure supplied with ASMXREF, is *userid*.XREFOUT.SOR.

On VSE the SOR is printed from SYSPRT.

The Spreadsheet Oriented report shows occurrences of specified tokens in the search library. Default tokens are specified in the XRFLANG file; tokens you have specified are contained in your XRFTOKN token file. For details on the default tokens see “ASMXREF XRFLANG Statements” on page 132. For details of the TOKEN control statements see “ASMXREF Token Statement” on page 128. The first record for each token set in the report is the “heading” record with the following quoted strings:

Member Name

Lines Of Code

Total Matches

token_1
⋮

token_n
⋮

The remaining records are detail records containing the module name, the number of lines of code, the total matches for the module, and the number of matches for each token.

The report segments each header and detail record in 80 byte segments, with the last segment having an EBCDIC CR (carriage return) and a LF (line feed) character. The report pads each record with spaces to fill the 80 characters. The report specifies the carriage return and line feed characters with a ◄.

Set title delimiters and cell separator characters with the report parameters. For details on the report parameters see “ASMXREP Options” on page 135.

When you run the ASMXREF scan phase, for the SOR report, ASMXREF generates the TSP (Tagged Source Program). The ASMXREF report phase uses the Tagged Source Program to create the SOR report.

```
'Module ID','LOC','Total Matches','*DATE','*YR','=C'20','=P'20','CSECT','DAT*',
'DATE','DAY','DD/MM/YY','MM/DD/YY','MM/YY','MONTH','TIME','YEAR','YR','YY/MM/DD'
','YYDDD'

ASMTEST,0069,0007,0002,0000,0000,0000,0001,0004,0000,0000,0000,0000,0000,0000,00
00,0000,0000,0000,0000
```

Figure 46. Sample Spreadsheet Oriented Report for MVS and CMS

```
'Module ID','LOC','Total Matches','*DATE','*YR','=C'20','=P'20','CSECT','DAT*','
'DATE','DAY','DD/MM/YY','MM/DD/YY','MM/YY','MONTH','TIME','YEAR','YR','YY/MM/DD
','YYDDD'

ASMTEST,0069,0007,0002,0000,0000,0000,0001,0004,0000,0000,0000,0000,0000,0000,00
00,0000,0000,0000,0000

/+
/*
```

Figure 47. Sample Spreadsheet Oriented Report for VSE

Note: When you create the Spreadsheet Oriented report on VSE, it generates two records at the end of the file:

```
/+
/*
```

Before importing the file into a spreadsheet, delete these records.

File Transfer to PC

To transfer the Spreadsheet Oriented report to a PC use the following settings in your file transfer program:

File option	ASCII Text
File option	One to one character mapping
Record format	Fixed
Logical record format	80

Note: Do not specify **Carriage Return, Line Feed**. ASMXREF specifies these in the Spreadsheet Oriented data file.

Symbol Where Used (SWU) Report

The Symbol Where Used (SWU) report lists all symbols referenced within the source and type of reference.

On CMS ASMXREF creates the SWU in a file named *filename* OUTSWU A, where *filename* is the name of the control file.

On MVS, the default name for the SWU, defined in the procedure supplied with ASMXREF, is *userid*.XREFOUT.SWU.

On VSE the SWU is printed from SYSPRT.

The symbols can be variables or macros identified with the following flags:

Comparison ASMXREF recognizes the symbol is a comparison.

Definition ASMXREF recognizes that the symbol is declared in that particular module.

External Ref ASMXREF recognizes the symbol is an external reference.

Label ASMXREF recognizes the symbol as a label.

Macro ASMXREF recognizes the symbol as a macro call.

Parameter ASMXREF recognizes the symbol is a parameter.

Read ASMXREF recognizes that the symbol is used in expressions but does not name locations.

Write ASMXREF recognizes that the symbol name is used as the target of an operation.

\# Number.

```
Date: 07/09/2004     ASMXREF  V1.5.0  Symbol Where Used Report           Page    1
Time: 12:09:28        Symbol to Module Map - All Symbols (SYM)


Symbol
   Module                                                        Access         #

&SYS
   ASMTEST ASSEMBLE  ..........................................      D         1

A
   ASMTEST ASSEMBLE  ..........................................     P          1

ANY
   ASMTEST ASSEMBLE  ..........................................  R             1

ASMHSAVE
   ASMTEST ASSEMBLE  ..........................................  R   D         2

ASMTEST
   ASMTEST ASSEMBLE  .......................................... LR             6

BEGIN
   ASMTEST ASSEMBLE  .......................................... L              2

BIN
   ASMTEST ASSEMBLE  ..........................................     P          1

CHAIN
   ASMTEST ASSEMBLE  .......................................... L              1

DATWORK
   ASMTEST ASSEMBLE  ..........................................  R   D         2

DBLWORK
   ASMTEST ASSEMBLE  ..........................................  RW  D         4

DD
   ASMTEST ASSEMBLE  .......................................... L W            2

E
   ASMTEST ASSEMBLE  ..........................................     P          1

EXIT
   ASMTEST ASSEMBLE  .......................................... L              1

FREEMAIN
   ASMTEST ASSEMBLE  ..........................................  R     M       1

GETMAIN
   ASMTEST ASSEMBLE  ..........................................  R     M       1

L
   ASMTEST ASSEMBLE  ..........................................     P          1
-------------------------------------------------------------------------------
  C = Comparison       D = Definition    E = External Ref     K = Class   L = Label
  M = Macro/Func/Inc   O = Object        P = Parameter        R = Read    W = Write
```

Figure 48 (Part 1 of 2). Sample Symbol Where Used (SWU) Report

```
Date: 07/09/2004     ASMXREF  V1.5.0  Symbol Where Used Report           Page    2
Time: 12:09:28        Symbol to Module Map - All Symbols (SYM)


Symbol
   Module                                                      Access         #

LENWORK
   ASMTEST ASSEMBLE  ..................................... LR             3

LINKAGE
   ASMTEST ASSEMBLE  .....................................     P          2

LV
   ASMTEST ASSEMBLE  .....................................     P          2

MF
   ASMTEST ASSEMBLE  .....................................     P          2

MM
   ASMTEST ASSEMBLE  .....................................   W  D         2

PARMS
   ASMTEST ASSEMBLE  ..................................... LR             2

R
   ASMTEST ASSEMBLE  .....................................     P          2

RETURN
   ASMTEST ASSEMBLE  ..................................... L              1

R0
   ASMTEST ASSEMBLE  .....................................   W  D         3

R1
   ASMTEST ASSEMBLE  .....................................  RW  D         4

R10
   ASMTEST ASSEMBLE  .....................................      D         1

R11
   ASMTEST ASSEMBLE  .....................................  RW  D         3

R12
   ASMTEST ASSEMBLE  .....................................  RW  D         5

R13
   ASMTEST ASSEMBLE  .....................................  RW  D        11

R14
   ASMTEST ASSEMBLE  ..................................... LRW  D         4

R15
   ASMTEST ASSEMBLE  .....................................  RW  D         5
-------------------------------------------------------------------------------
  C = Comparison       D = Definition    E = External Ref     K = Class   L = Label
  M = Macro/Func/Inc   O = Object        P = Parameter        R = Read    W = Write
```

Figure 48 (Part 2 of 2). Sample Symbol Where Used (SWU) Report

```
Date: 07/09/2004     ASMXREF  V1.5.0  Symbol Where Used Report                                    Page    1
Time: 12:19:34        Symbol to Module Map - All Symbols (SYMC)

Symbol                          Module   Access     Module   Access     Module   Access     Module   Access     Module   Access
-------------------------------------------------------------------------------------------------------------------------------
&SYS                            ASMTEST       D
A                               ASMTEST      P
ANY                             ASMTEST   R
ASMHSAVE                        ASMTEST   R   D
ASMTEST                         ASMTEST  LR
BEGIN                           ASMTEST  L
BIN                             ASMTEST      P
CHAIN                           ASMTEST  L
DATWORK                         ASMTEST   R   D
DBLWORK                         ASMTEST   RW  D
DD                              ASMTEST  L W
E                               ASMTEST      P
EXIT                            ASMTEST  L
FREEMAIN                        ASMTEST   R     M
GETMAIN                         ASMTEST   R     M
L                               ASMTEST      P
LENWORK                         ASMTEST  LR
LINKAGE                         ASMTEST      P
LV                              ASMTEST      P
MF                              ASMTEST      P
MM                              ASMTEST    W  D
PARMS                           ASMTEST  LR
R                               ASMTEST      P
RETURN                          ASMTEST  L
R0                              ASMTEST    W  D
R1                              ASMTEST   RW  D
R10                             ASMTEST       D
R11                             ASMTEST   RW  D
R12                             ASMTEST   RW  D
R13                             ASMTEST   RW  D
R14                             ASMTEST  LRW  D
R15                             ASMTEST   RW  D
R2                              ASMTEST       D
R3                              ASMTEST   RW  D
R4                              ASMTEST       D
R5                              ASMTEST       D
R6                              ASMTEST       D
R7                              ASMTEST       D
R8                              ASMTEST       D
R9                              ASMTEST       D
SYSTEM                          ASMTEST      P
TIME                            ASMTEST   R     M
TIMEMFL                         ASMTEST  LR
TIMWORK                         ASMTEST   RW  D
WORKAREA                        ASMTEST  LR
WTO                             ASMTEST  LR     M
WTOMSG                          ASMTEST    W  D
YY                              ASMTEST  L W
-------------------------------------------------------------------------------------------------------------------------------
Access Keys: C=Comparison D=Definition E=External K=Class L=Label M=Macro O=Object P=Parameter R=Read W=Write
```

Figure 49. Sample SWU sorted via SYMC

Token Where Used (TWU) Report

The Token Where Used (TWU) report shows occurrences of tokens in the search library.

On CMS ASMXREF creates the TWU in a file named *filename* OUTTWU A, where *filename* is the name of the control file.

On MVS, the default name for the TWU, defined in the procedure supplied with ASMXREF, is *userid*.XREFOUT.TWU.

On VSE the TWU is printed from SYSPRT.

You can specify tokens in the XRFTOKN file, use the default tokens specified in the XRFLANG file, or use both. For details on the TOKEN control statements see “ASMXREF XRFLANG Statements” on page 132 and “ASMXREF Token Statement” on page 128.

When you run the ASMXREF scan phase for the TWU report, ASMXREF generates the Tagged Source Program (TSP). The ASMXREF report phase uses the Tagged Source Program to create the TWU report. For details of the TSP see “Tagged Source Program (TSP)” on page 155.

The TWU report shows:

- The total number of matches (occurrences), of the specified token.
- The number of lines of code (LOC) scanned.
- The number of lines with matches.

```
                  ASMXREF  V1.5.0 TOKEN WHERE USED REPORT         PAGE     1
                  ---------------------------------------

MODULE: ASMTEST                                Date: 07/09/2004 Time: 12:06:41
LANG  : ASM
MATCHES TOKEN
------- ------------------------------------------------------------------------
   0    '*C''19*'
   0    '*DATE'
   0    '*P''19*'
   0    '*YR''
   2    'DAT*'
   0    'DATE'
   0    'DAY'
   0    'DD/MM/YY'
   1    'MM/DD/YY'
   0    'MM/YY'
   0    'MONTH'
   2    'TIME'
   0    'YEAR'
   0    'YR'
   0    'YY/MM/DD'
   0    'YYDDD'

LOC:   90 TOKEN MATCHES:    5 NUMBER OF LINES WITH MATCHES:    5
-------------------------------------------------------------------------------
```

Figure 50. Sample Token Where Used (TWU) Report

Tagged Source Program (TSP)

When you run the ASMXREF scan phase for the TWU and SOR reports, ASMXREF generates the Tagged Source Program (TSP).

On CMS ASMXREF creates the TSP in a file named *filename* DATATWU A, where *filename* is the name of the control file.

On MVS, the default name for the TSP, defined in the procedure supplied with ASMXREF, is *userid*.TWU.TAGGED.FILE.

On VSE the default name for the TSP, defined in the sample JCL, is XRFTWU.

Note: ASMXREF creates the TSP in the same file for both the TWU and SOR reports.

The ASMXREF report phase uses the TSP to create the TWU and SOR reports. The TSP contains the original source code records interspersed with comment records in the syntax of the language of the source file. The comment records appear above the source line which identifies the token. The comments show the token string encountered and a cumulative count of the number of times the scan has found the token so far in the source file.

Unless you use the NOSEP statement to turn off the creation of the separators, ASMXREF creates separators when it generates the TSP. For details on the NOSEP statement see page 131. Producing separators allows the TSP to be split into individual members that you can use to replace or create macro or copy libraries. On MVS, the separators are in IEBUPDTE format:

```
./ ADD NAME=source_file_name
```

You can run the IEBUPDTE utility program with the TSP as input.

On CMS, the separators are in the format:

```
./ ADD NAME=source_file_name
```

You can run a REXX EXEC named ASMXSEP EXEC (supplied with ASMXREF) that splits the TSP into its component files.

On VSE, the TSP contains Catalog statements that you can use as input to a LIBR job, that splits the sequential file into members of a librarian sublibrary. The separator is in Librarian format:

```
CATALOG NAME=source_file_name.source_type REPLACE=YES
```

```
./ ADD NAME=ASMTEST
*ASMZXREF MODULE = ASMTEST           07/09/2004 12:06:41
*ASMZXREF LANG   = ASM
*ASMZXREF MATCHES =    1
*ASMZXREF 'DAT*'
ASMTEST TITLE '- SAMPLE ASSEMBLY LANGUAGE PROGRAM WHICH USES DATES'     00001000
* ********************************************************* *          00002000
*   LICENSED MATERIALS - PROPERTY OF IBM                    *          00003000
*                                                           *          00004000
*   5696-234                                                *          00005000
*                                                           *          00006000
*   (C) COPYRIGHT IBM CORP. 1975, 2004. ALL RIGHTS RESERVED. *          00007000
*                                                           *          00008000
*   US GOVERNMENT USERS RESTRICTED RIGHTS - USE,            *          00009000
*   DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP         *          00010000
*   SCHEDULE CONTRACT WITH IBM CORP.                        *          00011000
*                                                           *          00012000
* ********************************************************* *          00013000
********************************************************************** 00014000
* THE SAMPLE ASSEMBLER SOURCE IS INTENDED AS INPUT TO THE ASMXREF     * 00015000
* PROGRAM.                                                            * 00016000
* NO CLAIMS ARE MADE AS TO THE FUNCTIONAL VALIDITY OF THE ASSEMBLER   * 00017000
* CODE.                                                               * 00018000
********************************************************************** 00019000
ASMTEST CSECT ,                      REENTRANT HLASM                    00020000
ASMTEST RMODE ANY                    LET THIS RUN ANYWHERE              00021000
ASMTEST AMODE 31                                                        00022000
         SPACE 2                                                        00023000
         USING *,R15                  ADDRESSABILITY TO ENTRY CODE      00024000
         B     BEGIN                                                    00025000
         DC    C'ASMTEST.&SYSDATE..&SYSTIME'                            00026000
BEGIN    STM   R14,R12,12(R13)        SAVE CALLERS REGISTERS            00027000
         LR    R12,R15                SAVE PTR TO EXIT PARAMETER LIST   00028000
         DROP  R15                                                      00029000
         USING ASMTEST,R12                                              00030000
         LR    R11,R1                 SAVE PTR TO EXIT PARAMETER LIST   00031000
         USING PARMS,R11                                                00032000
         LR    R3,R13                 GRAB PTR TO CALLERS SAVE AREA     00033000
         LA    R0,LENWORK             LOAD LENGTH OF WORK AREA NEEDED   00034000
         GETMAIN R,LV=(0)             GET STORAGE, LENGTH IN R0         00035000
         LR    R13,R1                 POINT R13 AT SAVE AREA            00036000
         USING WORKAREA,R13                                             00037000
CHAIN    ST    R13,8(,R3)             CHAIN THE CALLER TO THE EXIT      00038000
         ST    R3,4(,R13)             CHAIN THE EXIT TO THE CALLER      00039000
         SPACE 2                                                        00040000
         XC    TIMWORK,TIMWORK        CLEAR THE DECK.                   00041000
         LA    0,TIMWORK                                                00042000
         LA    2,TIMEMFL                                                00043000
*ASMZXREF MATCHES =    1
*ASMZXREF 'TIME'
         TIME  BIN,(0),LINKAGE=SYSTEM,MF=(E,(2)) ASK THE TIME.          00044000
         UNPK  DBLWORK(3),DATWORK+1(2)  UNPACK THE IMPORTANT BIT        00045000
         OC    DBLWORK(2),=C'00'        TURN INTO CHARACTERS            00046000
         MVC   WTOMSG,DBLWORK           COPY THE DISPLAY FORMAT TO MSG  00047000
WTO      WTO   'XX IS THE TWO DIGIT YEAR.'                              00048000
WTOMSG   EQU   WTO+8,2                                                  00049000
         SPACE 1                                                        00050000
         MVI   MM/DD/YY,'MM/DD/YY'                                      00051000
EXIT     LR    R1,R13               ADDRESS OF WORK AREA (FOR FREE)     00052000
         L     R13,4(,R13)          UNCHAIN SAVE AREAS                  00053000
         LA    R0,LENWORK           LENGTH TO FREE                      00054000
         FREEMAIN R,LV=(0),A=(1)    LENGTH IN R0, ADDR IN R1            00055000
RETURN   LM    R14,R12,12(R13)      RESTORE CALLERS REGISTERS           00056000
         XR    R15,R15              SET THE RC TO ZERO                  00057000
         BR    R14                  RETURN TO CALLER                    00058000
         SPACE 1                                                        00059000
WORKAREA DSECT ,                                                        00060000
ASMHSAVE DS    9D                                                       00061000
TIMWORK  DS    D                                                        00062000
*ASMZXREF MATCHES =    2
*ASMZXREF 'DAT*'
DATWORK  DS    D                                                        00063000
DBLWORK  DS    D                                                        00064000
*ASMZXREF MATCHES =    2
*ASMZXREF 'TIME'
TIMEMFL  TIME  ,LINKAGE=SYSTEM,MF=L                                     00065000
```

Figure 51 (Part 1 of 2). Sample Tagged Source Program (TSP)

```
LENWORK  EQU   *-ASMHSAVE                                              00066000
&SYS     DS    D                                                       00067000
*ASMZXREF MATCHES =    1
*ASMZXREF 'MM/DD/YY'
MM/DD/YY DS    F                                                       00068000
         SPACE 1                                                       00069000
R0       EQU   0                                                       00070000
R1       EQU   1                                                       00071000
R2       EQU   2                                                       00072000
R3       EQU   3                                                       00073000
R4       EQU   4                                                       00074000
R5       EQU   5                                                       00075000
R6       EQU   6                                                       00076000
R7       EQU   7                                                       00077000
R8       EQU   8                                                       00078000
R9       EQU   9                                                       00079000
R10      EQU   10                                                      00080000
R11      EQU   11                                                      00081000
R12      EQU   12                                                      00082000
R13      EQU   13                                                      00083000
R14      EQU   14                                                      00084000
R15      EQU   15                                                      00085000
         SPACE 1                                                       00086000
PARMS    DSECT                                                         00087000
         DS    2F                                                      00088000
         SPACE 1                                                       00089000
*ASMZXREF SUMMARY TOTAL TOKENS  =   16
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF '*C''19*'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF '*DATE'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF '*P''19*'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF '*YR''
*ASMZXREF SUMMARY TOTAL MATCHES =    2
*ASMZXREF 'DAT*'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'DATE'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'DAY'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'DD/MM/YY'
*ASMZXREF SUMMARY TOTAL MATCHES =    1
*ASMZXREF 'MM/DD/YY'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'MM/YY'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'MONTH'
*ASMZXREF SUMMARY TOTAL MATCHES =    2
*ASMZXREF 'TIME'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'YEAR'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'YR'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'YY/MM/DD'
*ASMZXREF SUMMARY TOTAL MATCHES =    0
*ASMZXREF 'YYDDD'
*ASMZXREF SUMMARY TOTAL TOKEN MATCHES =    5
*ASMZXREF SUMMARY LOC =   90
*ASMZXREF SUMMARY END
         END   ASMTEST                                                 00090000
```

Figure 51 (Part 2 of 2). Sample Tagged Source Program (TSP)

ASMXREF Messages

ASMXREF creates a message file containing information about its processing and about any error conditions it detects. Unless otherwise overridden, the message file uses the following file naming conventions:

- For CMS - *filename* LIST *filemode*

  where *filename* is the name of the control file.
- For MVS - SYSPRINT
- For VSE - SYSLST

This section explains the ASMXREF message format and the messages you may receive.

The message format is as follows:

```
ASMZnnn msglevel message_text
```

ASMZnnn The error message number.

msglevel A letter indicating the severity level. The letter associated with a numerical MSGLEVEL code or return code, as described in Table 22

message_text The character string *message_text* denotes variable text with specific information such as a *filename*, *table_name* or *record number*.

Table 22. Message Level

Message Level	Return Code	Description	Details
I	0	Information	ASMXREF informs you of actions taken. You probably expect the action. These messages keep you informed of the program's progress.
W	4	Warning	An ASMXREF action was taken or a condition encountered that may not produce the correct results. The condition or action taken is given in the message.
E	8	Error	These errors are expected to result in incorrect data. For example, an INCLUDE control statement explicitly requests that a specific module be processed, but the module is not found in the library.
S	12	Severe error	These messages indicate errors that can effect the entire run, such as ASMXREF control statement syntax errors. No processing is done when this type of error is found.
T	16	Terminating error condition	ASMXREF terminates processing when this error occurs.

ASMXREF issues all messages whose severity is equal to or greater than the message level you specify with the `MSGLEVEL` parameter.

Message List

ASMZ003S *statement_type* **Overflow in** *module_name* **processing record** *record_number*

Explanation: This error message occurs when a table in an ASMXREF system module overflows.

1. Typically this happens when a source statement is longer than the default sizes (currently 50,000 characters) provided for the associated parameters. The easiest way to fix the overflow is to use the ITBSIZE and LOGSIZE parameters in the control file to specify a larger size. See page 127 for details on the PARM control statement. For example, for the ITB or Logical Statement Table overflows, specify a PARM control statement following the LIBRARY control statement in the control file as follows:

   ```
   LIBRARY LIB=TEST1,TYPE=CMS,LANGUAGE=ASM
           PARM ITBSIZE=100000
           PARM LOGSIZE=100000
   ```

   The size of the parameters is limited only by the amount of memory available.

2. This message could also occur because of a language mismatch. For example, if ASMXREF is scanning an assembler language program using the COBOL language processor, a table could overflow when ASMXREF is searching for the ending delimiter. In such cases, you must specify the correct language using the LANGUAGE parameter in the control file.

System Action: The ASMXREF run terminates

Programmer Response: Increase the table sizes by specifying the ITBSIZE and LOGSIZE parameters as detailed in step 1 above and re-run ASMXREF, or correct the LANGUAGE statement.

ASMZ006T OPEN failed for XRFTOKN file.

Explanation: The file containing the TOKEN information is not found when executing ASMXREF.

System Action: The ASMXREF run terminates.

Programmer Response: Check the file definition for XRFTOKN in the job, and re-run ASMXREF. If the problem persists, contact your IBM service representative.

ASMZ007W *module_name* **previously processed from Library** *library_name*

Explanation: (CMS only) The following Module was previously processed - *module_name*

ASMXREF found the name of a member that has already been processed. If you specified YES to **Process duplicate modules** when you run the SCAN phase, ASMXREF processes modules with duplicate names; otherwise, ASMXREF bypasses them.

System Action: The ASMXREF run continues.

Programmer Response: Warning message only.

ASMZ008T The LIBRARY statement with TYPE=SEQ must have one INCLUDE control statement with the module name specified.

Explanation: You can specify sequential files only once with the Include control statement with module name.

System Action: The ASMXREF run terminates.

Programmer Response: Specify include control statements for each source file you need scanned and re-run ASMXREF.

ASMZ012T Include, Exclude, & Option Control Statements must be preceded by a library control statement.

Explanation: A LIBRARY control statement has been omitted or misplaced.

System Action: The ASMXREF run terminates.

Programmer Response: Edit the ASMXREF control statements to correct the position of the LIBRARY statement and re-run ASMXREF.

ASMZ013T Control statement READ error.

Explanation: ASMXREF was unable to read the file containing the control statements.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system or job log(s) to determine why the control statement file could not be read. There may be system error messages indicating an open or read error on this file. Consult your Systems Programmer for assistance.

ASMZ016W Analysis error in record *record_number* **near column** *column_number***. The following line(s) were ignored:**

Explanation: This error might be because of an ASMXREF scan misinterpretation, or a syntax error. ASMXREF displays the records that were skipped and not included in the intermediate data files for report inclusion and calculations.

System Action: The ASMXREF run continues.

Programmer Response: Check the LIBRARY control statement to determine whether the language parameter has been specified correctly. If the problem cannot be determined please contact your IBM service representative.

ASMZ017T Unable to OPEN file *filename*

Explanation: ASMXREF could not open the file *filename*.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system or job log(s) to assist in resolving why the file could not be read. There may be system error messages indicating an open or read error on this file. Consult your Systems Programmer for assistance.

ASMZ023T *Modulename* **not found.**

Explanation: The following Module was not found - *Modulename*.

System Action: The ASMXREF scan terminates.

Programmer Response: The FSOPEN macro failed for CMS when opening a library file. Consult your systems programmer for problem resolution. If the problem persists contact your IBM service representative.

ASMZ028T OPEN failed for SYSPRINT.

Explanation: The report file defined by the FILEDEF/DD/DLBL statement for SYSPRINT was not found.

System Action: The ASMXREP run terminates with user abend code 016.

Programmer Response: Check the file definition for SYSPRINT in the job. If the problem persists, contact your IBM service representative.

ASMZ029T OPEN failed for SYSINDS.

Explanation: The intermediate data file is not found when executing ASMXREP.

System Action: The ASMXREP run terminates.

Programmer Response: Check the FILEDEF/DD/DLBL statement for SYSINDS in the job step.

ASMZ030T OPEN failed for SYSINOU file.

Explanation: The report file defined by the FILEDEF/DD/DLBL statement for SYSINOU not found.

System Action: The ASMXREP run terminates.

Programmer Response: Check the FILEDEF/DD/DLBL statement for SYSINOU in the job step.

ASMZ031T Memory allocation failed in ASMZTWUS.

Explanation: Memory allocation failed in module ASMZTWUS.

System Action: The ASMXREP run terminates with user abend code 018.

Programmer Response: Check the region size if running on MVS and partition size if running on VSE. For VM, check the storage defined for the job. Increase the storage size. If the problem persists, contact your IBM service representative.

ASMZ032T PUT failed for record in SYSINOU.

Explanation: The PUT macro failed.

System Action: The ASMXREP run terminates with user abend code 020.

Programmer Response: Examine the system job log(s) for associated messages. Consult your systems programmer to correct the problem. If the problem persists, contact your IBM service representative.

ASMZ033T TOKEN statement does not contain any parameters.

Explanation: A TOKEN statement has been detected which does not contain a keyword.

System Action: The ASMXREF run terminates.

Programmer Response: Complete the TOKEN statement in the XRFTOKN file and re-run ASMXREF.

ASMZ034T Error in user control statements. Processing terminated.

Explanation: There are errors in the ASMXREF control statements.

System Action: The ASMXREF run terminates.

Programmer Response: ASMXREF has detected invalid control statements. Correct the control statements and re-run ASMXREF.

ASMZ036T TOKEN statement contains an invalid keyword.

Explanation: The TOKEN statement in the XRFTOKN file allows only certain keywords. ASMXREF has detected an invalid keyword.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the TOKEN statement, in the XRFTOKN file, that contains an invalid keyword and re-run ASMXREF.

ASMZ037T Parsing error in TOKEN statement.

Explanation: A parsing error is detected when processing the XRFTOKN file.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the erroneous TOKEN statement in the XRFTOKN file, and re-run ASMXREF.

ASMZ038T File XRFTOKN contains an invalid statement.

Explanation: The XRFTOKN file can only contain the following type of information: either a comment line (starting with * in column one) or a TOKEN statement (starting with the TOKEN keyword).

System Action: The ASMXREF run terminates.

Programmer Response: Correct the XRFTOKN file to ensure that it contains only allowed data, and re-run ASMXREF.

ASMZ039T No end-delimiter found in TOKEN statement.

Explanation: A TOKEN statement must be specified in matching, enclosing delimiters. ASMXREF has detected a TOKEN statement which contains a start-delimiter but does not contain an end-delimiter.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the XRFTOKN file to ensure that all tokens have matching start and end-delimiters and then re-run ASMXREF.

ASMZ042T An INC= statement must precede an EXC= statement

Explanation: A TOKEN EXC= keyword has been specified before an INC= control statement.

Programmer Response: The ASMXREF run terminates.

System Action: Insert a TOKEN INC= control statement before the TOKEN EXC= statement.

ASMZ043T Exclude statements are only applicable to generically specified tokens.

Explanation: A TOKEN EXC= keyword has been specified for an explicit TOKEN INC= token (a TOKEN INC= statement without wildcards).

Programmer Response: The ASMXREF run terminates.

System Action: Remove the exclude statement in error, or make the TOKEN INC= statement generic (with wildcards).

ASMZ044T The *macro_name* macro failed in ASMXREF module *module_name*

Explanation: An ASMXREF internal operating system macro failed.

Programmer Response: The ASMXREF run terminates.

System Action: Examine the system and job log(s) for associated error messages. Consult your system programmer to determine whether the processing error is the result of external errors. If the problem is external rectify it and re-run ASMXREF; otherwise contact your IBM service representative.

ASMZ045T An ABEND occurred in ASMXREF processing and a diagnostic dump has been requested.

Explanation: An abnormal termination occurred during ASMXREF processing.

Programmer Response: The ASMXREF run terminates.

System Action: Examine the system and job log(s) for associated error messages. Consult with the system programmer to determine whether the processing error is the result of external errors. If the problem is external rectify it and re-run ASMXREF; otherwise contact your IBM service representative.

ASMZ046T End-delimiter is not the last character in the TOKEN statement.

Explanation: A character has been detected past a token's matching enclosing delimiters. No additional data is allowed past the end-delimiter.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the XRFTOKN file to ensure that the token is enclosed within delimiters and no additional data follows the delimiter, and re-run ASMXREF.

ASMZ048T At least one statement is required in file XRFTOKN.

Explanation: The XRFTOKN file must contain at least one record which can either be a comment line (starting with * in column one) or a TOKEN statement (starting with the TOKEN keyword).

System Action: The ASMXREF run terminates.

Programmer Response: Edit the XRFTOKN file to provide the requested information, and re-run ASMXREF.

ASMZ052T Report name not specified.

Explanation: A valid Report name must be specified.

System Action: The ASMXREP run terminates.

Programmer Response: Specify the report name and re-run ASMXREP.

ASMZ053T Incorrect Report name specified: *report***.**

Explanation: Report name must be one of the following: CF, LOC, MWU, SWU, TWU or SOR.

System Action: The ASMXREP run terminates.

Programmer Response: Correct the erroneous report and re-run ASMXREP.

ASMZ054T Error Reading *defaults_file_name* **DEFAULTS file.**

Explanation: An error occurred while reading the DEFAULTS file.

System Action: The ASMXREP run terminates.

Programmer Response: Verify the integrity of the DEFAULTS file and re-run ASMXREP.

ASMZ055T Cannot find *defaults_file_name* **DEFAULTS file on any accessed disk.**

Explanation: The DEFAULTS file could not be found.

System Action: The ASMXREP run terminates.

Programmer Response: Ensure the DEFAULTS file is accessible or that the DEFAULT file is specified and re-run ASMXREP.

ASMZ056T No STAE Exit will be taken due to errors encountered when 'STAE' was issued.

Explanation: The STAE macro issued to install abend trapping and error recovery failed with a non-zero return code.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system or job log(s) to determine whether system error messages were issued. Consult your systems programmer to assist in problem resolution. If the problem persists consult your IBM service representative.

ASMZ057T Unable to load *module_name*

Explanation: An ASMXREF internal component is unable to be dynamically loaded.

System Action: The ASMXREF run terminates.

Programmer Response: Consult your systems programmer to confirm the ASMXREF install is error free. Examine the system and job log(s) for associated error messages. If the problem persists please contact your IBM service representative.

ASMZ058T Incorrect CMSTypeFlag used: *cmstype_flag***.**

Explanation: The only valid CMSTypeFlag are: HT or RT.

System Action: The ASMXREP run terminates.

Programmer Response: Correct the erroneous CMSTypeFlag and re-run ASMXREP.

ASMZ062T Page Length Greater than 20 is required.

Explanation: A Page Length greater than 20 encountered.

System Action: The ASMXREP run terminates.

Programmer Response: Change the Page Length to be greater than 20 and re-run ASMXREP.

ASMZ066T First record implies language is *language_name*

Explanation: The language in the LANGUAGE control statement is unknown to ASMXREF.

System Action: The ASMXREF run terminates.

Programmer Response: ASMXREF is unable to determine the source language scanned. Change the control statement, LIBRARY LANGUAGE=*language_name*, to specify a language keyword defined in the XRFLANG file.

ASMZ067T Language Keyword *keyword* **not recognized. Use LANGUAGE Control statement to respecify Language.**

Explanation: An invalid LIBRARY LANGUAGE= control statement has been specified.

System Action: The ASMXREF run terminates.

Programmer Response: The specified language is not valid. Change the control statement to specify a valid language type.

ASMZ073E SWU report not supported for *language*

Explanation: The SWU report is not available for the language specified.

System Action: None, ASMXREF stops processing.

Programmer Response: Change the LIBRARY LANGUAGE= control statement to specify a language supported by the SWU report.

ASMZ074T Invalid Library Type *library_type*

Explanation: ASMXREF detected an invalid library type on the LIBRARY control statement.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the library control statement and correct the TYPE keyword value and re-run ASMXREF.

ASMZ075T Library Type *library_type* **not supported on this Operating System.**

Explanation: A library type is specified that is not appropriate for this operating system. For example: `PDS specified on CMS.`

System Action: The ASMXREF run terminates.

Programmer Response: Correct the LIBRARY TYPE keyword value to specify the correct TYPE value and re-run ASMXREF.

ASMZ076T Filetype not specified for *file_name* **in Library** *library_name*

Explanation: On CMS, each record of the source listing file must contain a valid filename and filetype. ASMXREF has detected a filename in the source list file that does not specify the filetype. If a filemode is not specified, ASMXREF uses the first file found in the standard CMS search sequence.

System Action: The ASMXREF run terminates.

Programmer Response: Ensure that the source list file on CMS contains a list of files with valid filetypes and re-run ASMXREF.

ASMZ078T No file names specified in Library *library_name*

Explanation: (CMS only) None of the filenames specified in the source list file can be found by ASMXREF.

System Action: The ASMXREF run terminates.

Programmer Response: None of the specified files were found in the search order. Ensure that the files included in the source list file are accessible to your CMS machine.

ASMZ079T *filetype* **is a Reserved Filetype.**

Explanation: (CMS only) *filetype* is a Reserved File Extension.

A reserved filetype has been used.

System Action: The ASMXREF run terminates.

Programmer Response: On CMS, confirm that the filetype of the files listed in the source list file are of a valid filetype. Correct the filetype, then re-run ASMXREF.

ASMZ080T Failure in allocating storage in Module *module_name*

Explanation: A GETMAIN/GETVIS macro invocation failed to allocate virtual storage.

System Action: The ASMXREF run terminates.

Programmer Response: Try increasing your region/storage size and re-run ASMXREF. If the problem persists contact your IBM service representative.

ASMZ081E Control statement syntax error near column *column_number*

Explanation: ASMXREF met an error while parsing the ASMXREF control statements. The message indicates the column and line position that caused the error.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the control statements for error. Correct the syntax of the statement in error and re-run ASMXREF.

ASMZ082T Failure to release storage in Module *module_name*

Explanation: A FREEMAIN/FREEVIS macro failed to release virtual storage.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system job log(s) for associated error messages. Consult your systems programmer for problem resolution. If the problem persists contact your IBM service representative.

ASMZ084W Parsing error. The following lines were ignored:

Explanation: The lines following the error were ignored.

System Action: The ASMXREF run continues.

Programmer Response: This warning message is issued when the ASMXREF parser encounters source code it cannot parse. The source lines in error are skipped.

ASMZ095T Input file *filename* **DATA** *rep* **missing for Report** *report***.**

Explanation: The required input file is missing for the requested report.

System Action: The ASMXREP run terminates.

Programmer Response: Specify the Report in the Input File and re-run ASMXREP.

ASMZ096T *file_name* **DATA** *report* **does not match** *report* **Required LRECL of** *valid_format***.**

Explanation: The file does not have a valid format. The following formats are valid: for Report = 'MWU' the LRECL-format must be "F 96" for Report = 'SWU' the LRECL-format must be "F 93" for Report = 'TWU' the LRECL-format must be "F 80" for Report = 'SOR' the LRECL-format must be "F 80".

System Action: The ASMXREP run terminates.

Programmer Response: Correct the error(s) for the DATA file and re-run ASMXREP.

ASMZ099W Unrecognized Character in record *record_number* **near column** *column_number*

Explanation: ASMXREF found an invalid character.

System Action: The ASMXREF run continues.

Programmer Response: This warning message identifies the column position and record of the character in error.

ASMZ100T Incorrect Sort Order used: *sort_order***.**

Explanation: Sort Order must be one of the following: MAC, MOD, PART or SYM.

System Action: The ASMXREP run terminates.

Programmer Response: Correct the erroneous Sort Order and re-run ASMXREP.

ASMZ101T Incorrect Sort Order for SWU Report: *sort_order***.**

Explanation: Sort Order for SWU Report must be one of the following: MOD or SYM.

System Action: The ASMXREP run terminates.

Programmer Response: Correct the erroneous Sort Order for SWU report and re-run ASMXREP.

ASMZ102T Incorrect Sort Order for MWU Report: *sort_order***.**

Explanation: Sort Order for MWU Report must be one of the following: MAC or PART.

System Action: The ASMXREP run terminates.

Programmer Response: Correct the erroneous Sort Order for MWU report and re-run ASMXREP.

ASMZ103T No STAE work area passed from supervisor. No retry possible.

Explanation: No retry possible.

System Action: The ASMXREF run terminates.

Programmer Response: ASMXREF is unable to recover from an ABEND condition. The supervisor should have allocated storage and passed this to the ASMXREF recovery routine. The supervisor is unable to allocate storage for this and the ASMXREF recovery fails. Examine the system job log(s) for associated messages and consult your systems programmer for problem resolution. If the problem persists, consult your IBM service representative.

ASMZ104T Symbol Where Used Table overflow. Use SWUSIZE= PARM to rectify.

Explanation: The SWU table has insufficient space.

System Action: The ASMXREF run terminates.

Programmer Response: Specify a PARM SWUSIZE=*nnnnn* parameter in the ASMXREF control statements. If the parameter is already specified increase the numeric value and re-run ASMXREF. If the problem persists contact your IBM service representative.

ASMZ106T Macro Where Used Table overflow. Use MWUSIZE= PARM to rectify.

Explanation: The MWU table has insufficient space.

System Action: The ASMXREF run terminates.

Programmer Response: Specify a PARM MWUSIZE=*nnnnn* parameter in the ASMXREF control statements. If the parameter is already specified increase the numeric value and re-run ASMXREF. If the problem persists contact your IBM service representative.

ASMZ111T Work Disk *workmode* **not a valid Disk Accessed in WRITE Mode.**

Explanation: The Work Disk is not a valid disk accessed in WRITE Mode.

System Action: The ASMXREP run terminates.

Programmer Response: Ensure the Work Disk is a valid Disk Accessed in WRITE Mode and re-run ASMXREP.

ASMZ112T Work Disk *workmode* **not accessed.**

Explanation: The Work Disk cannot be accessed.

System Action: The ASMXREP run terminates.

Programmer Response: Ensure the Work Disk is accessible. and re-run ASMXREP.

ASMZ113T Work Disk *workmode* **not accessed in WRITE Mode. Specify another Output Disk.**

Explanation: The Work Disk cannot be accessed in WRITE Mode.

System Action: The ASMXREP run terminates.

Programmer Response: Specify another Output Disk and re-run ASMXREP.

ASMZ116I ASMXRPT completed with Return Code = *exitrc***.**

Explanation: The ASMXREP program terminated with a Return Code.

System Action: The ASMXREP run terminates.

Programmer Response: Check the value of the Return Code.

ASMZ118T Sort Order not required for Report *report***.**

Explanation: The ASMXREP program terminated with a Return Code.

System Action: The ASMXREP run terminates.

Programmer Response: Check the Sort Order is correct for the report specified.

ASMZ120I TSP File *fn ft fm* **is being processed.'**

Explanation: (CMS only) The TSP file *fn ft fm* is being processed.

System Action: None.

Programmer Response: None.

ASMZ122T CNTLMode *cntlmode* **is Incorrect.**

Explanation: The disk mode specified for the CNTL file is in correct.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the parm and re-run ASMXREF.

ASMZ124I Default sort order SYM used for SWU reports.

Explanation: No sort parameter passed to ASMVRPT.

System Action: The ASMXRPT run continues.

Programmer Response: The default sort sequence is used. Refer to the parameters option to change the sort sequence. To suppress this message, supply the default sort sequence as a parameter.

ASMZ125T This library cannot be processed because of problems reading SDDS(CLEAR) or directory (PDS).

Explanation: ASMXREF has encountered problems with a PDS directory.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system job log(s) for associated messages. Consult your systems programmer for problem resolution. If the problem persists contact your IBM service representative.

ASMZ126T Dynamic allocation failed for library *library_name*

Explanation: ASMXREF is unable to find the library *library_name*.

System Action: The ASMXREF run terminates.

Programmer Response: Ensure that the library specified in the ASMXREF control statement is accessible and specified correctly. If the library is accessible and no other associated system or job log messages are issued, consult your IBM service representative.

ASMZ127I Default sort order MAC used for MWU reports.

Explanation: No sort parameter passed to ASMVRPT.

System Action: The ASMXRPT run continues.

Programmer Response: The default sort sequence is used. Refer to the parameters option to change the sort sequence. To suppress this message, supply the default sort sequence as a parameter.

ASMZ134T *filename file_type file_mode* **must have RECFM = F and LRECL = 80.**

Explanation: The attribute of the above mentioned file must have the following attributes: RECFM = F and LRECL = 80.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the attributes for RECFM/LRECL and re-run ASMXREF.

ASMZ135T *filename* **CNTL** * **not found.**

Explanation: The file referenced above could not be found.

System Action: The ASMXREF run terminates.

Programmer Response: Ensure the CNTL file is present and re-run ASMXREF.

ASMZ136T Work Disk *work_mode* **is Incorrect.**

Explanation: A problem with the Work Disk encountered.

System Action: The ASMXREF run terminates.

Programmer Response: Check for the integrity of the Work Disk and re-run ASMXREF.

ASMZ137T Work Disk *work_mode* **is not Accessed in Write Mode.**

Explanation: The Work Disk cannot be accessed in WRITE Mode.

System Action: The ASMXREF run terminates.

Programmer Response: Ensure the Work Disk is accessible in Write Mode and re-run ASMXREF.

ASMZ138T ASMXSEP completed with Return Code = *rc***.**

Explanation: (CMS only) The TSP file has been processed by ASMXSEP procedure.

System Action: Check the return code for any processing errors.

Programmer Response: Re-run the process if needed.

ASMZ140W *module_name* **was not found in library.**

Explanation: The file *module_name* is either empty or not found.

An INCLUDE control statement named a module not in the input source library.

System Action: ASMXREF continues.

Programmer Response: The included modules (source files) were either empty or do not exist in the source library. Ensure that the included names are correct and re-run ASMXREF.

ASMZ141T Message Level must be a numeric value between 0 and 16.

Explanation: The Message Level parameter contains an erroneous value.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the erroneous Message Level value and re-run ASMXREF.

ASMZ142T Incorrect Duplicates parm used: *duplicates***.**

Explanation: The Duplicate parameter specified is incorrect.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the erroneous Duplicate parm and rerun ASMXREF.

ASMZ143T Page Length Field must contain a numeric value.

Explanation: An erroneous value for the Page Length Field was coded.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the erroneous Page Length Field and re-run ASMXREF.

ASMZ144T Page Length must be between 20 and 999.

Explanation: An erroneous Page Length was coded.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the Page Length Field and re-run ASMXREF.

ASMZ145T *module_name* **is empty or file not found.**

Explanation: The following file is either empty or not found - *module_name*

This module does not exist, or an I/O error occurred during the search.

System Action: The ASMXREF run terminates.

Programmer Response: Examine the system job log(s) for associated messages. Consult your systems programmer and correct the problem. If the problem persists contact your IBM service representative.

ASMZ146W Message limit exceeded. No more X-level messages will be printed

Explanation: More than 60 messages of severity level "X" have been issued. All further messages of the same severity level are suppressed

System Action: The ASMXREF run continues.

Programmer Response: The default message limit is 60. For diagnosis specify a message level of 1 to allow all messages to be printed re-run ASMXREF.

ASMZ149T Incorrect ReturnMsg parm used: *return msg***.**

Explanation: An erroneous ReturnMsg parm was used.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the ReturnMsg parm to be 'YES' or 'NO' and re-run ASMXREF.

ASMZ165T Syntax error in *report_id* **report parm field.**

Explanation: No data for the indicated report is produced.

System Action: The ASMXREF run terminates.

Programmer Response: Correct the erroneous PARM field for the indicated report and re-run ASMXREF.

ASMZ167W Empty Library *library_name*

Explanation: This error occurs most often on MVS when a PDS library is empty.

System Action: The ASMXREF run continues.

Programmer Response: Ensure that the LIBRARY control statement has specified the correct PDS. The user can use system utilities to verify the content of the PDS. If the PDS has been specified correctly and has valid members, contact your IBM service representative.

ASMZ169T Token exceeds system limit of *nn* **characters.**

Explanation: The ASMXREF control statement parser has detected a token greater then the limit of *nn* characters.

System Action: The ASMXREF run terminates.

Programmer Response: Shorten the incorrect token to less than the limit of *nn* characters, and re-run ASMXREF.

ASMZ170T Exclude tokens may not be specified for non-generic scan tokens.

Explanation: The ASMXREF control statement parser has detected a TOKEN EXCLUDE statement specified with a mask character (wildcard).

System Action: The ASMXREF run terminates.

Programmer Response: TOKEN EXCLUDE statements must not contain a mask character. Remove the mask character from the incorrect exclude token statement, or remove the entire exclude token statement, and re-run ASMXREF.

ASMZ171I The TOKEN NODEFLT option is in effect. There will be no default token list processing.

Explanation: The default token list processing has been turned off with the TOKEN NODEFLT statement in the XRFTOKN input file. Confirm that this option is intended.

System Action: None, ASMXREF continues.

Programmer Response: None.

ASMZ172I No DEFAULT tokens were found in the XRFLANG file.

Explanation: DEFAULT TOKEN header not found in XRFLANG input file. Confirm that this option is intended.

System Action: None, ASMXREF continues.

Programmer Response: None.

ASMZ173T No DEFAULT tokens were found in the XRFLANG file and there are no TOKEN statements specified in XRFTOKN.

Explanation: No TOKEN statements were found in XRFTOKN file and no default tokens were specified in XRFLANG file.

System Action: None, ASMXREF terminates.

Programmer Response: Either add TOKEN statements into the XRFTOKN file, remove the TOKEN NODEFLT statement, or add DEFAULT TOKENS in the XRFLANG file, and re-run ASMXREF.

ASMZ174T The TOKEN delimiters were not matched in the MASK token: *token_in_error*

Explanation: Tokens must be specified with matching enclosing delimiters.

System Action: The ASMXREF run terminates.

Programmer Response: Enclose the TOKEN statement in matching delimiters, and re-run ASMXREF.

ASMZ175T The token MASK statement is in error: *token_mask*

Explanation: The token MASK statement must be specified in matching enclosing delimiters. The mask delimiter character cannot be longer than one character.

System Action: The ASMXREF run terminates.

Programmer Response: Edit the incorrect token mask, and re-run ASMXREF.

ASMZ176I No matching LANG= header found in the XRFLANG file for this run.

Explanation: The XRFLANG file does not have a language header for the one specified in the LIBRARY LANGUAGE= statement specified for this run. ASMXREF uses the default of ASM (assembler).

System Action: None, ASMXREF continues. Confirm this is correct.

Programmer Response: None.

ASMZ177T TOKEN NODEFLT is specified without any TOKEN INC/EXC in XRFTOKN file.

Explanation: The XRFTOKN file does not have any tokens and processing of the default tokens has been turned off with the TOKEN NODEFLT statement.

System Action: None, ASMXREF terminates.

Programmer Response: Enter token statements in the XRFTOKN file or remove the TOKEN NODEFLT statement from the XRFTOKN file.

ASMZ178T Unable to open XRFLANG file.

Explanation: The XRFLANG file is not available.

System Action: The ASMXREF run terminates.

Programmer Response: Ensure job control includes the XRFLANG file definition.

ASMZ179T ASMZXREP encountered array index out of bounds in ASMZTWUS.

Explanation: Array index is out of bounds.

System Action: The ASMXREP run terminates.

Programmer Response: Report this error condition to your IBM service representative.

ASMZ180T ASMZXREP encountered an incorrect file format in ASMZTWUS.

Explanation: The TWU file format is incorrect.

System Action: The ASMXREP run terminates.

Programmer Response: Report this error condition to your IBM service representative.

ASMZ181T ASMZXREP encountered an incorrect file format ASMZTWUS.

Explanation: The TWU file format is incorrect.

System Action: The ASMXREP run terminates.

Programmer Response: Report this error condition to your IBM service representative.

ASMZ182I MEMTYPE control card not specified. The default of A is used.

Explanation: For VSE users, MEMTYPE control card is need for processing, if not included it defaults to 'A'.

System Action: The ASMXREF run continues.

Programmer Response: None.

ASMZ183I LANGUAGE control card not specified. The default of ASM is used.

Explanation: The LIBRARY LANGUAGE control card is not specified. ASMXREF defaults to LIBRARY LANGUAGE=ASM

System Action: The ASMXREF run continues.

Programmer Response: None.

ASMZ184I Default sort order MOD used for LOC reports.

Explanation: No sort parameter passed to ASMVRPT.

System Action: The ASMXRPT run continues.

Programmer Response: The default sort sequence is used. Refer to the parameters option to change the sort sequence. To suppress this message, supply the default sort sequence as a parameter.

ASMZ185I Default sort order MAC used for CFC reports.

Explanation: No sort parameter passed to ASMVRPT.

System Action: The ASMXRPT run continues.

Programmer Response: The default sort sequence is used. Refer to the parameters option to change the sort sequence. To suppress this message, supply the default sort sequence as a parameter.g

ASMXREF User Abends

Table 23. ASMXREF Abend Codes

Code	Message
003	ASMXREF is unable to open the SYSIN file
004	ASMXREF is unable to open the SYSPRINT file
005	A 005 abend can occur either from the ASMXREF message processing module or XRFMSG when a virtual storage request failed
006	ASMXREF encountered an unrecognized symbol when it scanned an input module
009	ASMXREF encountered a logical statement whose length is outside the acceptable range
010	Buffer record length is less than the left source margin

Table 24. ASMXREP Abend Codes

Code	Message
016	ASMZ028T OPEN failed for SYSPRINT
018	ASMZ031T Memory allocation failed in ASMTWURS
020	ASMZ032T PUT failed for record in SYSINOU
022	ASMZ030T OPEN failed for SYSINOU

Chapter 6. Using Enhanced SuperC

Introduction to Enhanced SuperC 174
The SuperC Comparison 174
The SuperC Search 175
SuperC Features for Date Comparisons 176
General Applications 176
How SuperC and Search-For Filter Input File Lines 177
How SuperC Corrects False Matches 178
How SuperC Partitions and Processes Large Files 179
Comparing Load Modules 179
Comparing CSECTs 179
Invoking the SuperC Comparison 180
Invoking the Comparison on MVS 180
MVS JCL Example 180
Invoking the Comparison on CMS using Menu Input 183
COMMAND 184
New File ID and Old File ID 184
Member 185
Selection List 185
Compare Type 186
Listing Type 186
Listing File ID 187
Process Options 188
Process Statements ID 189
Update File ID 190
Display Output 190
Auto Display Pgm 191
Primary Comparison Menu PF Key Definitions 191
Printing the Wide Listing 192
Invoking the Comparison on CMS using Command Line Input 192
Types of Options 193
Command Line Priority and Overriding 198
Compares From FILELIST 198
Invoking the Comparison on VSE 199
VSE JCL Example 1: Non-VSAM-managed Sequential Files 199
VSE JCL Example 2: VSAM-managed Sequential Files 202
VSE JCL Example 3: VSAM Files 203
VSE JCL Example 4: Tape Files 203
VSE JCL Example 5: Librarian Members 204
Invoking the SuperC Search 205
Invoking the Search on MVS 205
MVS JCL Example 205
Invoking the Search on CMS using Menu Input 207
Invoking the Search on CMS using Command Line Input 215
Types of Options 217
Command Line Priority and Overriding 221
SRCH Process Statement Directive 221
Invoking the Search on VSE 221
VSE JCL Example 1: Non-VSAM-managed Sequential Files 222
VSE JCL Example 2: VSAM-managed Sequential Files 223
VSE JCL Example 3: VSAM Files 224

© Copyright IBM Corp. 1992, 2004

VSE JCL Example 4: Tape File . . . 224
VSE JCL Example 5: Librarian Members . . . 225
Process Options . . . 226
Process Statements . . . 236
Change Listing Value . . . 238
Change Text . . . 238
Comment Lines . . . 240
Compare Byte Offsets . . . 241
Compare (Search) Columns . . . 241
Compare Lines . . . 242
Compare Sections . . . 244
DD-MVS Alternate DD Names . . . 245
DD-VSE DLBL/TLBL Definitions . . . 246
Define Column Headings . . . 249
Do Not Process Lines . . . 250
Exclude Data . . . 252
Focus on Data . . . 253
Line Count . . . 254
List Columns . . . 254
List Previous-Search-Following Value . . . 254
Revision Code Reference . . . 255
Search Strings in the Input File . . . 255
Select Files from a List of Files (CMS) . . . 257
Select Members or Files (CMS) . . . 258
Select Members (VSE) . . . 259
Select PDS Members (MVS) . . . 260
Statements File Listing Control . . . 261
Title Alternative Listing . . . 262
Work Size . . . 262
Year Aging . . . 263
Date Definitions . . . 263
Global Date . . . 266
CMS Command Line Option Directives . . . 266
CMS Command Line Statement Option Directives . . . 267
Understanding the Listings . . . 269
General Listing Format . . . 269
How to View the Listing Output . . . 269
The Comparison Listing . . . 270
Page Headings . . . 270
Listing Output Section . . . 271
Member Summary Section (CMS) . . . 273
Overall Summary Section . . . 275
Examples of Comparison Listings . . . 276
The Search Listing . . . 283
Page Heading . . . 283
Source Lines Section . . . 284
Summary Section . . . 287
Examples of Search Listings . . . 288
Update Files . . . 292
Revision File . . . 293
Revision File (2) . . . 294
Update CMS Sequenced 8 File . . . 295
Update Control Files . . . 296
Update Long Control . . . 299

Update MVS Sequenced 8 File 300
Update Prefixed Delta Lines 301
Update Sequenced 0 File 301
Update Summary Only Files 302
CMS File Selection List 305
Getting to the Selection List Menus 306
The Selection List Menu (Comparison) 306
The Selection List Menu (Search) 308
How SuperC Pairs CMS Files and Members 311
CMS Files Used by SuperC 312
Reasons for Differing Comparison Results 313
Return Codes 314
SuperC Messages 316

Introduction to Enhanced SuperC

The comparison and search facility (named Enhanced SuperC and referred to in the rest of this chapter as SuperC) is a versatile program that can be used to compare two sets of data (using the SuperC Comparison) or to search a specific set of data for a nominated search string (using the SuperC Search).

SuperC is designed to run on the following platforms:

- MVS (batch)
- VM/ESA (CMS menu or CMS command line interface)
- VSE/ESA (batch)

At a minimum, the SuperC Comparison requires only the names of the two items to be compared. The SuperC Search requires only the name of the item to be searched and the search string.

You can tailor the comparison or search using *process options* and *process statements*. Process options are single keywords that you enter on the PARM parameter (MVS and VSE), a menu (CMS), or the command line (CMS). Process statements consist of a keyword and one or more operands; you pass these to SuperC in an input file.

For example, you can use the *process option* ANYC (“Any Case”) so that SuperC treats uppercase and lowercase characters as the same. (Thus, “d” and “D” are considered to be the same.) You can use the *process statement* DPLINE (“Do not Process Lines”) to ignore the lines (being compared or searched) that contain a specified character string. For example, `DPLINE '$'` causes all lines that contain the single-character string “$” to be ignored.

The SuperC Comparison

Using the SuperC Comparison, you can:

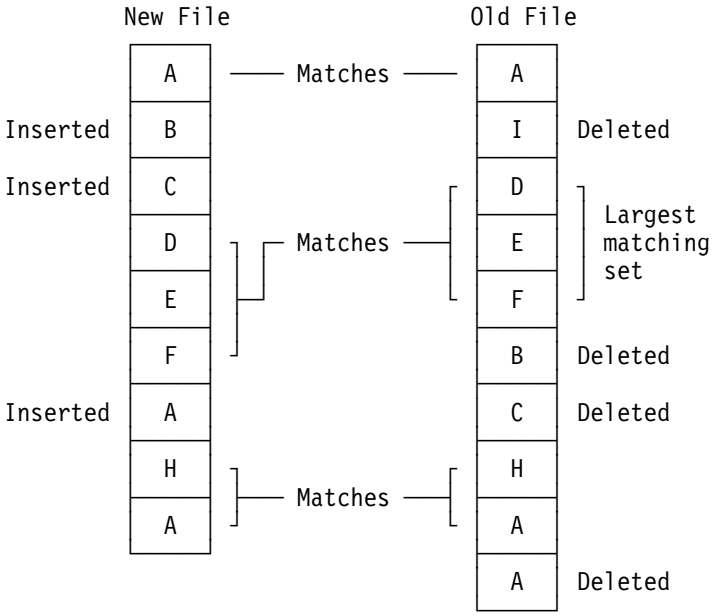
- Specify at what “level” the comparison is to be performed (file, line, word or byte)
- Exclude certain data from the comparison
- Restrict the comparison to certain types of data
- Handle various date formats (for example, 2-digit and 4-digit year representations)
- Control the type of listing output produced
- Specify an update file to be produced

SuperC operates independently of any synchronization data, such as column or sequence numbers. It does not use the common “start at the top then look ahead or look back” method to determine large sections of matching data. Neither does it sort the data before comparing. SuperC is unique in that, except for files that are identical, no match determination is made until both files have been completely read.

SuperC recognizes matching and missing files, lines, words, or bytes (data units) based on data content only. “Missing” data units are units that are out of sequence, as opposed to units that have been deleted from a file. It finds all matches, locates

the largest set of matching data units, and recursively allows this *comparison set* to divide the file into additional partitioned subsections. All new subsections are processed for corresponding matches. The sub-process ends when no more matches can be found within corresponding *new* and *old* file partitioned subsections. Sections classified as “inserted” or “deleted” are corresponding areas for which no matches were found.

Figure 52 demonstrates how SuperC compares two files which have records (“lines”) represented by A, B, C, ... The SuperC algorithm attempts to find the best match set from the input records. Notice how the match set requires consideration of duplicate lines.



Comparison Sequence	New File	Result	Old File
Largest set	D E F	Matches set	D E F
Top set	A	Matches	A
Leftover top set	B C	Mismatches	I
Largest bottom match	H A	Matches	H A
Leftover bottom set	A	Mismatches	B C A

Note: The inserted “A” on the lower left cannot connect with the deleted “A” on the bottom right due to the “H” and “A” barrier.

Figure 52. Illustration of How SuperC Compares Files

The SuperC Search

Using the SuperC Search, you can specify:

- One or more search strings
- Whether multiple search strings are independent of each other or must be present on the same line
- Whether a search string is a word, prefix, or suffix
- The range of columns to be searched

- The number of lines to appear in the output listing before and after each line where a search string is found

SuperC Features for Date Comparisons

Using SuperC features specifically designed to help you manage dates, you can:

- Specify a 100-year period (or “year window”) so that, for dates that have only a 2-digit year, the century can be determined. This can be based on either:
  - A “fixed” year window (with a fixed starting year), or
  - A “sliding” year window (starting at a specified number of years prior to the current year).
- Compare 2-digit year values in one file with 4-digit year values in another file.
- Compare compressed year values in one file with uncompressed year values in another file.
- Filter cosmetic differences caused by adding century digits to 2-digit years, so that you can more easily identify real differences in content.

General Applications

SuperC provides many features for general applications and all types of users.

General users can:

- Compare two files that have been reformatted. Reformatted files contain such differences as indentation level changes, or inserted or deleted spaces.

  SuperC detects and classifies reformatted lines as special changes. You can list these lines in the output, along with the normal insert/delete changes, or eliminate them from the listing. Reducing the number of flagged lines may help focus on real, rather than cosmetic, changes.
- Determine whether two *groups* of files have corresponding like-named “components.”

  Components absent from one group but present in the other are listed, as is all change activity between like-named components. The comparison can show changes caused by creating or deleting components of file groups.

Writers and editors can:

- Detect word changes within documents.

  SuperC finds word differences even if the words have been moved to adjacent lines.
- Verify that only designated areas are changed.

  SuperC comparison results show all areas affected. Changes made to restricted areas may be invalid. Unintended changes can therefore be detected so that a complete document need not be checked for errors again.
- Use SuperC to automatically insert SCRIPT/VS or IBM BookMaster® revision codes.

  The UPDREV process option can be used with either the WORD or LINE compare type to put either SCRIPT/VS (.rc) or IBM BookMaster (:rev and :erev) tags before and after the changed lines.

Programmers and systems administrators can:

- Generate management reports that show the quantity and type of changes in program source code.

  SuperC can count the changed and unchanged lines of code in an application program. Comparison results could be used, for example, to summarize the changes between different versions of a program.

- Retain a record of change activity.

  SuperC listing files can be collected and retained as a permanent record of the changes made before a new program is released. Source code differences can help detect regressions or validate the appropriateness of any code modifications.

- Modify a listing output file, including additional headers or change delimiters.

  Some SuperC listings may need to be rewritten before you accept the results. For example, some installations may require security classifications. Others may require a listing created using the WIDE process option to have box delimiters surrounding changed sections.

- Compare files across unconnected systems.

  SuperC can generate a 32-bit hashsum per file using the FILE compare type. Files compared on an unconnected processor, using SuperC, should have the same hashsums if they are identical. A FILE comparison on any file to determine a hashsum can be done by specifying the same file as both *new* and *old*.

- Develop additional uses for update files.

  SuperC produces general results with generalized listings. However, your installation may have unique requirements. There are many specialized update files that you can use to produce listings that match these requirements. Normal SuperC listings may not fit this type of application, but the update files are more structured and should be easier to use as data input. See “Update Files” on page 292 for explanations and examples of the update files.

How SuperC and Search-For Filter Input File Lines

The SuperC and Search-For utilities apply process options and process statements to the input file or files in a specific order. Figure 53 on page 178 shows schematically the effects, in the order that they occur, of the various “filtering” process options and process statements, on the compare and Search-For input lines. The options and statements nearer the top affect the input line before options or statements nearer the bottom.

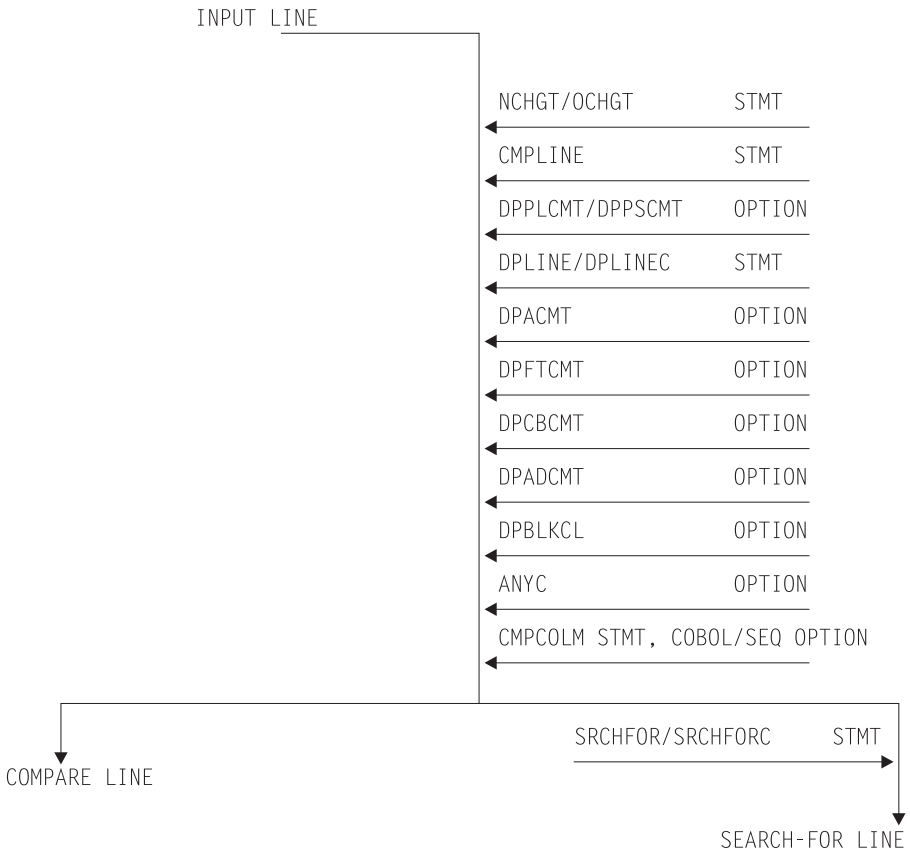


Figure 53. Priority for Filtering Input Lines

How SuperC Corrects False Matches

Occasionally, SuperC reports that it has detected a false line or word match and has corrected the results in the listing and summary report. Any affected matched pair has been reclassified as an insert/delete pair. Any resulting error might be in the masking of potential matches that would be overlooked due to the early false match coupling. That is, an equivalent yet undiscovered match may be overlooked due to the premature false matching. The condition should be of minor importance since it happens so rarely and the masking effect has a low probability of affecting the final results.

An equally important SuperC concern would be whether it finds the best match set and whether it finds all matches. Unfortunately, the match-finding algorithm is not perfect. Ignoring the false match masking problem, and the large number of duplicate source lines obscuring the match set possibilities, occasional matches can be overlooked. SuperC , however, does not fail to correctly classify mismatches and does not incorrectly classify a mismatch as a match.

Comparison of large files, can sometimes lead to false matches. Increasing the WORKSIZE process statement value can sometimes alleviate the number of false matches reported.

How SuperC Partitions and Processes Large Files

In SuperC, there is no limit on the size of files processed in terms of lines, words or bytes. Yet it has an internal methodology based upon a maximum field size for each work area storage structure. SuperC performs the overall comparison process by breaking very large files into smaller comparison partitions and combining the intermediate results into one overall result. The process attempts to ensure that the file partitioning does not appear to be determined after some arbitrary limit is reached. This can affect the results on either side of the break point.

A partitioning size of 32000 lines/words/bytes is the default. This size can be adjusted by specifying the WORKSIZE process statement. The compare processes up to this limit and iteratively adjusts the intermediate ending break point of the pass by an adaptive method. Continuation from the adjusted end point is the basis for the next pass. That end point might even be adjusted to some previous records that have already been processed. The objective is to achieve the next best compare set for future unprocessed records.

The overall process ends when both files reach the End-of-File during a pass. The results from the intermediate passes are combined into one user end result. Most large compares never appear to have been partitioned and recombined.

Comparing Load Modules

SuperC compare of load module data might show unexpected differences. This is because SuperC compares all the data in the load module as it is found on DASD, and does not attempt to decode which portions are executable, and which might contain uninitialized storage.

The complex data format on DASD is dependent on the load module data set block size, and defined storage definitions which are controlled by the linkage editor. The size stored by the linkage editor in the PDS directory may differ from the DASD data byte count reported by SuperC and Browse depending on the characteristics of the load module.

If load modules are exact copies of each other, SuperC should find no differences. If load modules have been link-edited from the same object but with different blocksizes, SuperC will probably report they are different.

Because of the relative DASD addresses (TTRs) in load modules, the recommended procedure for comparing load modules which have not been reblocked is to use the AMBLIST utility with LISTLOAD OUTPUT=MODLIST against both load modules, then use SuperC to compare the two AMBLIST outputs. There is no easy way to compare load modules with different internal record sizes such as occurs when COPYMOD or LINKEDIT processes them.

Comparing CSECTs

SuperC compare of PDS Load Module CSECTs (using the LMCSFC Process Option) can return unexpected differences. SuperC looks at the length of the CSECT from the control record immediately preceding the CSECT data record in the load module. This physical data length can differ from the logical CSECT data

length in the load module header that the AMBLIST utility uses to report the length of the CSECT.

SuperC always compares all of the physical data in each CSECT. You can use SuperC Byte compare to examine the CSECT data content in detail.

Note: This option is only valid for PDS load modules.

Invoking the SuperC Comparison

The following sections describe how to invoke the SuperC Comparison on each platform (MVS, CMS and VSE).

Invoking the Comparison on MVS

On MVS, you invoke the SuperC Comparison as a batch program. You can use the SuperC Comparison on MVS to compare:

- Two sequential data sets
- Two complete partitioned data sets
- Two VSAM data sets
- Members of two partitioned data sets
- Concatenated data sets
- A VSAM data set with a sequential data set

MVS JCL Example

Figure 54 on page 181 shows simplified MVS JCL to run the SuperC Comparison. This example is supplied with SuperC in the sample PDS (default is ASM.JMQ415A.SASMSAM2) as member ASMFMVC1.

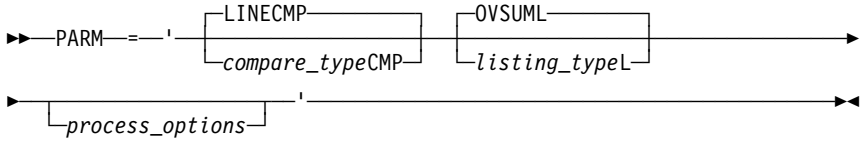
Before running this example, edit the lines highlighted by numbers (such as **1**) as described in the instructions following the example listing.

```
  ⋮
/*
/* Run the comparison with these options (see 1 and 6 )
/*
//RUN      EXEC PGM=ASMFSUPC,REGION=4M,PARM='options'                    1
/*
//STEPLIB   DD   DSN=#hlq.SASMMOD2,DISP=SHR                              2
/*                                                                       ┐
/* Define "new" data set to be compared                                  │
/*                                                                       │
//NEWDD    DD   DSN=new_file,DISP=SHR                                   ├ 3
/*                                                                       │
/* Define "old" data set to be compared                                  │
/*                                                                       │
//OLDDD    DD   DSN=old_file,DISP=SHR                                   ┘
/*
/* Direct listing data set to SYSOUT
/*
//OUTDD    DD   SYSOUT=*                                                 4
/*
/* Define update ("delta") data set
/*
//DELDD    DD   DSN=update_file                                          5
//SYSIN    DD   *
process_statements                                                       6
  ⋮
/*
//
```

Figure 54. Sample MVS JCL to Run the SuperC Comparison

1 Replace `PARM='options'` with a PARM parameter in the following format:

Format of ASMFSUPC PARM parameter for SuperC Comparison



Note: Each option may be separated by either a space or a comma.

compare_type
The type of comparison you want performed: FILE, LINE, WORD, or BYTE. When specifying the compare type in the PARM parameter, add the suffix “CMP” (for example, WORD becomes WORDCMP).

For a description of each compare type, see “Compare Type” on page 186.

listing_type
The type of listing you want from the comparison: OVSUM, DELTA, CHNG, LONG, or NOLIST. When specifying the listing type in the PARM parameter, add the suffix “L” (for example, CHNG becomes CHNGL).

For a description of each listing type, see “Listing Type” on page 186.

process_options
Process options are keywords that direct SuperC how to perform the comparison or format the listing. Process options can be separated by spaces or commas.

For a description of each process option, see “Process Options” on page 226.

For example:

```
PARM='LINECMP DPCBCMT DELTAL NOSUMS'
```

instructs SuperC to:

- Perform a line-by-line comparison. (LINE compare type with “CMP” suffix.)
- Ignore COBOL comment lines. (Process option DPCBCMT ignores lines with an “*” in column 7.)
- Produce a listing showing changes, without an overall summary section. (Process option NOSUMS eliminates the group and final summary listing from the output listing.)

2 Replace *#hlq* with the high level qualifier where SuperC is installed (default load library is ASM.JMQ415A.SASMMOD2).

3 Replace *new_file* and *old_file* with the items to be compared. These can be:

- Two sequential data sets
- Two complete partitioned data sets
- Two VSAM data sets
- Members of two partitioned data sets
- Concatenated data sets
- A VSAM data set with a sequential data set

Note: The terms “new” and “old” are used only for the sake of identifying the files being compared, which might or might not be different versions of the same file.

If you specify partitioned data set (PDS) names for *new_file* and *old_file*, SuperC compares all members in the new PDS with any like-named members in the old PDS. Members in either PDS not having like-named members in the other data set are not compared, but are reported in the listing data set.

To restrict a comparison of partitioned data sets to selected members only, use the SELECT process statement. For example, the following process statement:

```
SELECT NEW1:OLD1,SAME
```

instructs SuperC to compare only:

- Member NEW1 in the new PDS with member OLD1 in the old PDS
- Member SAME in the new PDS with member SAME in the old PDS

For more information on the SELECT process statement, see “Select PDS Members (MVS)” on page 260.

4 The listing data set, listing the results of the comparison. For example listings, see “Understanding the Listings” on page 269.

5 (Required only if you specify a “UPD...” process option; see **1** .)

The update (or “delta”) data set. Most update data sets are intended to be used as input to other tools, rather than being “human-readable” reports (such as the listing data set; see **4**). For instance, if you specify the UPDMVS8 process option, SuperC creates an update data set that you can use with the IEBUPDTE utility. You can use IEBUPDTE to apply to the old data set any updates that SuperC found in the new data set.

The file attributes of the update data set depend on the “UPD...” process option you specified.

For more information on producing an update data set, see the process options whose keywords start with “UPD” on Table 25 on page 226. For a selection of sample update data sets, see “Update Files” on page 292.

6 Insert any process statements (one statement per line) that you want to use.

For example, the following process statements:

```
CMPCOLM 7:72
LSTCOLM 1:72
```

instruct SuperC to compare only columns 7 to 72 in the new and old data sets (for example, if you want to compare COBOL source without comparing sequence numbers), but to include in the listing data set columns 1 to 72 (that is, the listing *contains* the sequence numbers).

For more information on process statements, see “Process Statements” on page 236.

Invoking the Comparison on CMS using Menu Input

You can use the SuperC Comparison on CMS to compare:

- Two files
- Selected files within file groups
- Two complete file groups
- Selected members within MACLIBs or TXTLIBs
- Complete MACLIBs or TXTLIBs

To invoke the SuperC Comparison on CMS, enter:

```
ASMFSUPC
```

on the CMS command line.

If you enter `ASMFSUPC` without any parameters, the Primary Comparison Menu appears (see Figure 55 on page 184). This menu allows to specify the files to be compared, and other SuperC options. However, if you enter `ASMFSUPC` with the file IDs to be compared and any options you want to use, the comparison starts immediately without displaying the menu.

This section describes how to use the Primary Comparison Menu. For information on invoking the SuperC Comparison directly from the command line without using the menu, see “Invoking the Comparison on CMS using Command Line Input” on page 192).

```
HLASM Toolkit Feature SuperC Compare Program V1R5M0 - Primary Menu
COMMAND ==>

                         Fn    Ft    Fm               (MACLIB/TXTLIB Files Only)
New File ID        ==>                          Member       ==>
Old File ID        ==>                          Member       ==>
                         Optional Section
Selection List     ==> NO                       ( NO / * )

Compare Type       ==> LINE                     (FILE/ LINE /WORD/BYTE)
Listing Type       ==> DELTA                    (OVSUM/ DELTA /CHNG/LONG/NOLIST
Listing File ID    ==> SuperC LIST A            (file-ID/ newfn SuperC A )

Process Options    ==>
                   ==>
Process Stmts ID   ==>                          ( file-ID               )

Update  File  ID   ==>
Display Output     ==> YES                      (YES/NO/COND/UPD        )
Auto Display Pgm   ==> XEDIT                    (BROWSE/XEDIT/EPDF/etc.  )

 1=Help  3/4=Quit  5=Proc Stmts  6=SrchFor  8=Proc Opts  9=Print  ENTER/10=Exec
```

Figure 55. SuperC Primary Comparison Menu

For a straightforward comparison of two files, just enter the names of the two files that you want compared. One is called the *new* file, the other the *old* file. (They are assumed to be different versions of the same file; the significance of “new” and “old” is normally irrelevant.) Enter the IDs (*fn ft fm*) of the two files in the **New File ID** and **Old File ID** fields respectively.

For example, if you want to compare the file NEW TEST1 A with the file OLD TEST1 A, enter the two file names as follows:

```
New File ID   ==> new test1 a
Old File ID   ==> old test1 a
```

and press Enter.

Here is more information about the other input fields on the SuperC Primary Comparison Menu. Default field values are underlined.

COMMAND

Use this field to issue CP and CMS commands, such as FILELIST, ERASE, or RDRLIST.

New File ID and Old File ID

The names of the two files to be compared. SuperC supports the CMS convention of including wildcard characters (“*”) and equal signs (“=”) as part of the input file ID.

This example compares NEW TEST1 A with OLD TEST1 A:

```
New File ID   ==> new test1 a
Old File ID   ==> old =     =
```

Other examples of file name usage are:

File ID Specified	Meaning
`new test1 a`	Single CMS file
`new test* a`	File group (all with a filetype starting with “TEST”)
`new maclib`	The entire macro library, NEW

Notes:

1. If a process statements file is specified (see “Process Statements ID” on page 189) and it contains a SELECTF process statement, the New File ID and Old File ID fields are ignored.
2. A MACLIB/TXTLIB with a filename containing an “*” (for example, ABC* MACLIB A or * TXTLIB C) is not processed as individual MACLIB/TXTLIBs with members. There is no method for specifying the “concatenation” of more than one MACLIB/TXTLIB.
3. The percent wildcard character (“%”) is not supported by SuperC.
4. SuperC allows the same file ID to be entered in both the New File ID and Old File ID fields. You can use SuperC in this way to obtain:
   - Various file statistics (at either the FILE, LINE, WORD or BYTE level)
   - A file hex dump listing (using a BYTE comparison with a LONG listing)
   - A comparison of different columns or rows within the same file.

Member

The name of the member, within either a macro library (MACLIB) or text library (TXTLIB), to be compared. (This field is only used when the file specified in New File ID refers to a macro or text library.) If left blank, all members within the specified library are selected for the comparison.

File ID Specified	Member	Meaning
`new maclib c`	`xyz`	XYZ member in NEW MACLIB C.
`new maclib c`	`*`	All members in NEW MACLIB C. (Selection List must = NO)

Selection List

Indicates if the Selection List facility is to be used. Valid values are:

NO Selection list facility not required.

***** Selection list facility required.

Note: The Selection List facility is only applicable when an “*” (asterisk) is contained within either the New File ID or the Old File ID. (In the case of a macro or text library, an “*” must be contained within the specified Member name.)

Enter an “*” in the Selection List field to see a list of files from which you can select the ones that you want.

The following examples explain the files that are listed for selection according to the file ID specified:

File ID Specified	Member	Files Listed for Selection
`new test1 *`		All files with the filename “NEW” and the filetype “TEST1”
`old test* a`		All files with the filename “OLD” and a filetype beginning with “TEST” and filemode “A”
`new txtlib a`	`*`	All members within the text library NEW TXTLIB A
`new maclib a`	`abc*`	All members within the macro library NEW MACLIB A whose name begins with “ABC”

For information on using the selection list, see “CMS File Selection List” on page 305.

Compare Type

The type of comparison to be performed. Valid values are:

FILE Compares source file for differences, but does not show what the differences are. This is the simplest and fastest method, with the least amount of processing overhead. For this compare type, SuperC reports only summary information.

LINE Compares source files for line differences. It is the most commonly-used compare type (and the default). The output report lists inserted and deleted lines; changed lines are treated as a deletion and insertion. Line lengths may be of any size.

Unequal record lengths are padded with spaces. There are no other padding options. A compare type of LINE informs you whether the data content is the same or not. It is common to compare lines from two files, ignoring the sequence columns in 73-80. However, this may yield results that differ from when a compare type of FILE is used (see “Reasons for Differing Comparison Results” on page 313).

WORD Breaks the files into lines and then into individual words. The results are similar to those for the LINE compare type except words on adjacent lines can be matched.

Word delimiters are normally spaces and end-of-line. The XWDCMP process option lets you use the standard set of non-alphanumeric characters in addition to spaces as delimiters.

BYTE Compares source files for byte differences. The output listing files consists of a hexadecimal printout with character equivalents listed on the right. The summary listing at the end details the number of bytes processed in the comparison.

To obtain a complete hex dump of a file, compare the file against itself, specifying a BYTE compare type with a LONG listing type.

Listing Type

The type of listing output required.

(For a detailed explanation of the format and content of the various listings produced by SuperC, see “Understanding the Listings” on page 269.)

Valid values are:

OVSUM Lists only an overall summary of the results of the comparison without showing the differences themselves. A group comparison generates an individual summary line for each file (or member) in the group. For more information, see “Overall Summary Section” on page 275.

DELTA Lists only the differences between the source files or members being compared, followed by overall summary results. Differences are flagged in the listing output section of the SuperC listing. For more information, see “Listing Output Section” on page 271.

For example, a DELTA listing of a LINE comparison shows only the individual lines in each file or member that are different.

CHNG Contains the same information as the DELTA listing, plus up to 1000 matching lines (default is 10) before and after the differences. This listing shows differences within the context of the surrounding matching data. To specify the number of lines shown before and after each difference, use the CHNGV process statement. For more information, see “Change Listing Value” on page 238.

LONG Lists the entire file, indicating where the differences exist, followed by a summary section.

NOLIST Produces no listing output. One of these messages is displayed on the menu:

```
Differences were found.
```

or

```
No differences were found.
```

Listing File ID

The name of the listing file generated as a result of the comparison. (A listing file is always generated unless the NOLIST listing type is specified.)

You can:

- Leave this field blank (in which case SuperC allocates a default name for the listing file)
- Specify a full file ID to be used for the listing file
- Use a combination of “*” and “=” symbols (which results in the listing file ID being a combination of the *fn ft fm* specified in the New File ID and the details you enter for the Listing File ID)

Here are some examples:

New file ID	Listing file ID	File ID Used
new test a		new superc a
new test a	myname mytype a	myname mytype a
* test a		$ superc a
new test a	= listing a	new listing a
new* test a	* listing a	new$ listing a

Process Options

You can specify the process options that you want (if any) by:

- Entering them directly using one (or both) of the process option lines on the Primary Comparison Menu

or

- Selecting them from the Process Options Selection Menu (PF8)

For a full list and description of process options, see “Process Options” on page 226.

Entering Process Options Directly: Type in each process option keyword on an entry line (each keyword must be separated by a space). Each line holds up to 51 characters (including spaces).

```
HLASM Toolkit Feature SuperC Compare Program V1R5M0 - Primary Menu
COMMAND ==>

                         Fn    Ft    Fm         (MACLIB/TXTLIB Files Only)

  ⋮

Process Options    ==>  locs anyc
                   ==>

  ⋮

 1=Help  3/4=Quit  5=Proc Stmts  6=SrchFor  8=Proc Opts  9=Print  ENTER/1
```

Figure 56. SuperC Primary Comparison Menu with Process Options Entered Directly

Figure 56 shows two process options entered directly on the Process Options line:

LOCS (“List Only Changed Entries in Summary”)
ANYC (“Any Case”)

Selecting Process Options from a Menu: You can also specify process options by selecting them from the Process Options Selection Menu. The menu shows the process options that are valid for the specified comparison type. To display the menu, press PF8.

For instance, if you are using a LINE compare type, pressing PF8 displays all of the process options for a LINE comparison (see Figure 57 on page 189).

```
HLASM Toolkit Feature SuperC Compare Program - Line-Compare Options    (1 of 4)
COMMAND ==>
Select option(s) from the following list or "blank" to remove.

Sel                        General Process Options
      SEQ      - Ignore sequence columns 73-80 on F 80 input source files.
      NOSEQ    - Process columns 73-80 as data on F 80 input source files.
      COBOL    - Ignore sequence columns 1-6 on F 80 input source files.
 S    LOCS     - List only changed and non-paired entries in group summary list.
      REFMOVR  - Reformat override.  Don't flag reformatted lines in listing.
      DLREFM   - Don't list reformatted old file lines. Only new file reformats.
 S    ANYC     - Process text lines as upper case.

                           Listing Process Options
      WIDE     - Up to 80 columns side-by-side.  Line length = 202/203.
      NARROW   - Up to 55 columns side-by-side.  Line length = 132/133.
      LONGLN   - Lists up to 176 columns.  Line length = 202/203.
      GWCBL    - Generate Word/Line Change Bar Listing.
      NOPRTCC  - No print control and page separators.
      ERASRC0  - Erase listing on compare return code = 0.

                                                      ( cont'd )

 PF1=Help            PF3=Menu            PF7=Prev Page           PF8=Next Page
```

Figure 57. Example of the SuperC Process Options Selection Menu (LINE Comparison)

To select a process option, enter an “S” next to it.

Process options which have been selected previously appear with an “S” alongside them (as for LOCS and ANYC in Figure 57).

When you no longer need a process option, clear the “S” from the Process Option Selection Menu or delete the option keyword from the Process Option field on the Primary Compare Menu.

Process Statements ID

The name of the file (if any) containing process statements.

Process statements (which are similar to process options but require one or more additional items of information to be specified) are always passed to SuperC in a file.

For a full list and description of process statements, see “Process Statements” on page 236.

You can either enter the name of an existing file that contains process statements, or press PF5 to create a new file and specify the process statements.

Pressing PF5 displays the Process Statements Entry Menu (see Figure 58 on page 190) showing examples of some of the process statements available and allows you to enter (one at a time) the process statements that you want.

When you exit from the Process Statements Entry Menu, SuperC automatically generates a file (called SUPERC SYSIN A) containing each of the process statements you specified. (`SUPERC SYSIN A` is entered against Process Stmts ID on the Primary Comparison Menu.)

```
HLASM Toolkit Feature SuperC Compare Program - Process Statements      (1 of 1)
Process Statements  ------  SuperC Compare Program  ------------------


Enter Process Statements for Statements File:
   ==>

     Examples                              Explanation

  CMPCOLM  5:60  75:90        Compare using two column compare ranges
  LSTCOLM  25:90              List columns 25:90 from input files

  ⋮

 PF1=Help    PF3=Menu    PF5=Menu     PF6=Cancel      ENTER=Save Line
```

Figure 58. Example of the SuperC Process Statements Entry Menu (Comparison)

Note: When you press PF5, SuperC erases any existing SUPERC SYSIN A file before creating the new file.

Update File ID

The name of the update file generated (if applicable) as a result of the comparison. SuperC generates an update file if you specify one of the “UPD...” process options.

You can:

- Leave this field blank (in which case SuperC allocates a default name for the update file)
- Specify a full file ID to be used for the update file
- Use a combination of “*” and “=” symbols (which results in the update file ID being a combination of the filename specified in the New File ID and the details you enter for the Update File ID

Here are some examples:

New File ID	Update File ID	File ID Used
new test a		new update a
new test a	myname mytype a	myname mytype a
* test a		$ update a
new test a	= updseq a	new updseq a
new* test a	* updseq a	new$ updseq a

Note: For further information, see “Update Files” on page 292.

Display Output

This field determines if the results of the comparison are to be displayed. Valid entries are:

YES Display the output listing using the editor specified in the Auto Display Pgm field.

NO Do not display any output.

COND Display the output listing using the editor specified in the Auto Display Pgm field if the return code is not 0 (that is, differences *were* found.)

UPD Display the update listing using the editor specified in the Auto Display Pgm field. UPD is only valid when an "UPD..." process option is specified in the Process Options field.

(space) Do not display any input unless the Auto Display Pgm field (see following description) is not space, in which case the Display Output field defaults to YES.

Auto Display Pgm

This field is used in conjunction with the Display Output option. It allows you to use the editor or browse program of your choice.

Specify the name of an editor or browse program to inspect the output listing:

program name The name of a valid editor or browse program to be invoked to display the results of the comparison. (For example, *XEDIT*, *EPDF*, *BROWSE*)

(space) Defaults to XEDIT if the Display Output option is *YES, COND or UPD*.

Primary Comparison Menu PF Key Definitions

PF1 Help. Displays the Help Table of Contents menu.

PF3 Quit. Leaves the current SuperC environment. SuperC terminates.

PF4 Quit. Same as PF3 from this menu. SuperC terminates.

PF5 Proc Stmts (Process Statements). Displays the Process Statements Entry Menu. This menu contains examples of the more widely-used process statements. It also has a field that allows you to input one process statement at a time into the SUPERC SYSIN A file.

Note: When you press PF5, SuperC erases any existing SUPERC SYSIN A file before creating the new file.

PF6 Displays the Primary Search Menu.

PF8 Proc Opts (Process Options). Displays the Process Options Selection Menu. The actual menu that is displayed depends on the contents of the compare type field. For example, in Figure 55 on page 184, pressing PF8 displays the first of three LINE process option menus.

PF9 Print. Builds a command to schedule the printing of the listing file. This command is then displayed in the command line area.

Note: If SuperC finds differences in the comparison process and you have used the WIDE process option (see page 235), the command displayed in the command line area causes the Wide Print Menu to be displayed (see Figure 59 on page 192) after you press Enter.

PF10 Execute and Quit. Verifies user-input fields, invokes SuperC, and quits.

ENTER Execute. Verifies user-input fields and invokes SuperC. Control returns to the Primary Comparison Menu after the comparison has completed.

Printing the Wide Listing

If you used the WIDE process option and SuperC finds differences in the comparison, pressing the PF9 key from the Primary Comparison Menu, followed by Enter, displays the Wide Print Menu (see Figure 59).

This menu displays the listing file ID and the printer information that you last specified (or the printer information in SUPERC NAMES *) allowing you to change these details if necessary.

Note: For a description of the SUPERC NAMES * file, see "CMS Files Used by SuperC" on page 312.

For an example of a side-by-side WIDE listing, see Figure 84 on page 280.

```
HLASM Toolkit Feature SuperC Compare Program - Wide Print Menu 06/11/2004
COMMAND ==>

Wide Print Listing of file:  MYTEST LISTING A1


Enter/Verify 3800 Printer Information (Defaults from SuperC Names file):
Printer Model    ==>  3800
        Spool    ==>  DIST 400-9999 CL 5 GT 15 FORM 3860 FCB FCB6
          Tag    ==>  RALVMX SYSTEM 20


For NOPRTCC Wide listings:
Lines per Page   ==>  0               (0-99, default=0 -- no page ejects)


New printer information is stored in LASTING GLOBALV if  ENTER is pressed.

PF1=Help          PF3/PF6=Primary Menu            PF4=Quit           ENTER=Exec
```

Figure 59. Example of the SuperC Wide Print Menu

Invoking the Comparison on CMS using Command Line Input

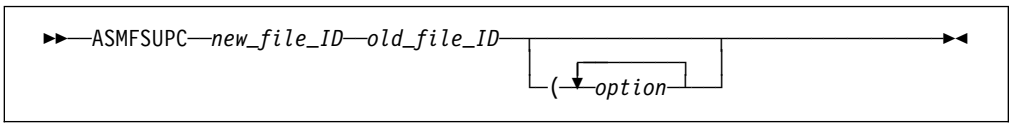
You can use the SuperC Comparison on CMS to compare:

- Two files
- Selected files within file groups
- Two complete file groups
- Selected members within MACLIBs or TXTLIBs
- Complete MACLIBs or TXTLIBs

This section describes how to invoke the SuperC Comparison directly from the command line, without using the Primary Comparison Menu. For information on using this menu, see "Invoking the Comparison on CMS using Menu Input" on page 183.

To invoke the SuperC Comparison from the CMS command line, enter `ASMFSUPC` with the file IDs to be compared and any options you want to use. The comparison starts immediately without displaying the Primary Comparison Menu.

The general format is:



new_file_ID The name of the new file (or member)

old_file_ID The name of the old file (or member)

option Each type of option is described in the following pages.

For example, to compare file TEST1 NEW A with file TEST1 OLD A (without specifying any options), enter:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A
```

To compare file TEST1 NEW A with file TEST1 OLD A (with a listing type of DELTA and a process option of WIDE), enter:

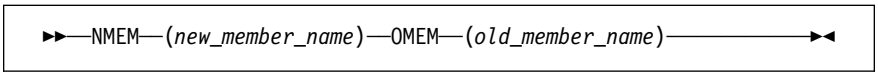
```
ASMFSUPC TEST1 NEW A TEST1 OLD A (DELTA WIDE
```

Types of Options

You can specify any of the following options in the CMS command line or in the Options List file (see page 197):

Member Names

This option specifies the names of the members within a library.



For example:

```
ASMFSUPC MACLIB NEW A MACLIB OLD A (NMEM(ABC) OMEM(DEF)
```

compares the member ABC in MACLIB NEW A with the member DEF in MACLIB OLD A.

Note: Member names can only be used as options when the *new_file_ID* and *old_file_ID* specified refer to either macro or text libraries.

Compare Type

This option specifies the type of comparison to be performed.

Can be *one* of the following keywords:

FILE	File comparison
LINE	Line comparison
WORD	Word comparison
BYTE	Byte comparison

For further descriptions of each compare type, see page 186.

Listing Type

This option specifies the type of listing output required.

Can be *one* of the following keywords:

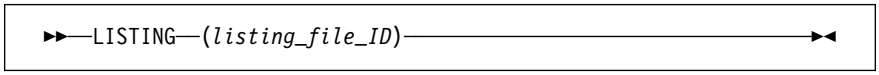
OVSUM	Overall summary
DELTA	Differences only
CHNG	Lines before/after differences

LONG Entire file
NOLIST No listing output

For further descriptions of each listing type, see page 186.

Listing File

This option specifies the alternative name to be assigned to the listing file generated as a result of the comparison process.



For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (LISTING(TSTLIST RESULTS A)
```

creates a listing file named TSTLIST RESULTS A.

Notes:

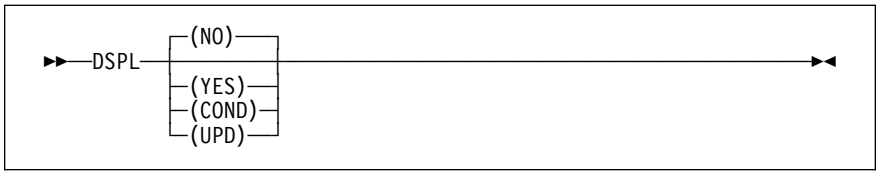
1. If you do not use the LISTING option, the listing file is generated with a default ID consisting of:

   fn Filename of the *new* file
   ft SUPERC
   fm A

2. A listing file is always generated unless the NOLIST listing type is specified.

Display Output

This option specifies if the results of the comparison are to be displayed.



NO Do not display output

YES Display output

COND Display output if differences found

UPD Display differences if update option used

For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (DSPL(YES)
```

causes the comparison results to be displayed.

Note: If you specify an editor or browse program (see following option), the “Display Output” option defaults to YES.

For a further description of the Display Output option, see “Display Output” on page 190.

Auto Display Program

This option is used in conjunction with the Display Output option. It allows you to use the editor or browse program of your choice (provided

that it is supported in your processing environment). The default is XEDIT.

Examples:

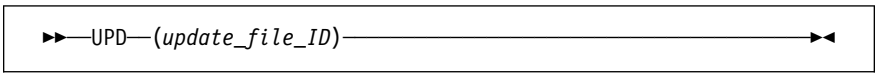
```
ASMFSUPC TEST1 NEW A TEST1 OLD A (EPDF

ASMFSUPC TEST1 NEW A TEST1 OLD A (XEDIT

ASMFSUPC TEST1 NEW A TEST1 OLD A (BROWSE
```

The above examples specify editors EPDF and XEDIT, and browse program BROWSE, respectively.

For a further description of the Auto Display Program, see “Auto Display Pgm” on page 191.

Update File ID

This option specifies the alternative name to be assigned to the update file generated (if applicable) as a result of the comparison process.



For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (UPD(TSTUPD DETAILS A)
```

creates an update file named TSTUPD DETAILS A.

Notes:

1. An update file is only generated if one of the “UPD...” process options was specified. For further details, see “Process Options” on page 226.
2. If you do not use the UPD option, the update file is generated with a default ID consisting of:

   fn Filename of the *new* file
   ft UPDATE
   fm A

Process Options

This option specifies the process options to be used in the comparison process.

These can be one or more of the process option keywords which are valid for the compare type used. For details of these, see “Process Options” on page 226.

For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (ANYC
```

specifies the process option ANYC with the result that the *case* of characters in the two input files is ignored when performing the comparison process.

Option Directives

You can use any of the following option directive keywords:

ERASRC0 Erase listing file if no differences
MENU Display Primary Comparison Menu

NOIMSG No information messages
NONAMES No SUPERC NAMES * file
NOOLF No Options List file
PRINT Print results

For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (NOOLF
```

specifies that any options contained in the Options List file are not to be used in the comparison process.

For further descriptions of each Option Directive, see "CMS Command Line Option Directives" on page 266.

Process Statement Directives

The following directives are transformed into process statements. They can be *one* of the following keywords:

CC Compare columns
LC List columns
LT Line count
RR Revision code reference

For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (LC(7:14)
```

selects columns 7 to 14 to be listed in the output.

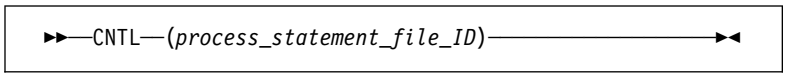
For further descriptions of each Process Statement Directive, see "CMS Command Line Statement Option Directives" on page 267.

Process Statement ID

This option specifies how process statements are to be supplied to the SuperC Comparison.

One of two keywords can be used:

CNTL Use the CNTL keyword if you want to use an existing file that contains the process statements you require.



For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (CNTL(TSTPRO OPTS A)
```

specifies that the process statements in file TSTPRO OPTS A are to be used.

PROMPT

PROMPT indicates to SuperC that the file SUPERC SYSIN A is to be used to supply the process statements and causes the Process Statements Entry Menu to be displayed. This menu contains examples of the more widely-used process statements. It also has a field that allows you to input one process statement at a time into the SUPERC SYSIN A file.

For example:

```
ASMFSUPC TEST1 NEW A TEST1 OLD A (PROMPT
```

causes the Process Statements Entry Menu to be displayed.

Note: If SUPERC SYSIN A already exists, its contents are erased before creating the new file.

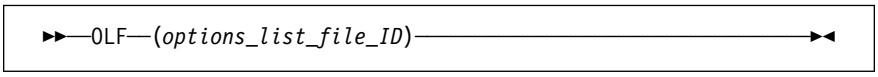
Options List File

You can use the Options List file to hold a set of default options (to save you entering them each time on the CMS command line). Any of the options described in this section can be placed in the Option List file. They take effect unless overridden by options in the command line.

If you do not specify a name for the Options List file, SuperC looks for a file with the default name SUPERC OLIST A and, if found, uses the options contained in that file for the comparison process.

However, you can nominate an alternative Options List file by using the keyword OLF. OLF allows you to specify either a fully-qualified file ID (*fn ft fm*) or a partially-qualified file ID for the Options List file that you want SuperC to use (see “Default Naming Convention for Options List File” on page 197).

Note: SuperC uses options contained in an OLF-specified Options List file before those in SUPERC OLIST A (see “Command Line Priority and Overriding” on page 198).



For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') OLF(MYOPTS FILE A)
```

specifies that the options in file MYOPTS FILE A are to be used.

Note: Not all options in the Options List file can be overridden since there is no way to negate them. Take care when considering which options to include in the file when using OLF.

To examine this further, let's look at an example of an Options List file containing the following:

```
DELTA XEDIT CNTL(MYFILE STMTS A)
```

If the Options List file that you nominate in the CMS command line (by using the OLF keyword) contains the above options, you *can*:

- Override the DELTA option by specifying any of the other listing types (for example, NOLIST) in the command line.
- Nullify the XEDIT Auto Display Program by including the Display Output option DSPL(NO) in the command line.

but you *cannot* override the Process Statements ID keyword CNTL (and therefore the process statements contained in the file MYFILE STMTS A take effect).

Default Naming Convention for Options List File: The command line uses the following defaults in the naming of the Options List file:

Command Line	OLF ID Used
ASMFSUPC...(...	SUPERC OLIST A
ASMFSUPC...(NOOLF...	(none)
ASMFSUPC...(OLF(TST1)...	TST1 OLIST A
ASMFSUPC...(OLF(TST1 OPTS)...	TST1 OPTS A
ASMFSUPC...(OLF(TST1 OPTS A)...	TST1 OPTS A

Command Line Priority and Overriding

The following priority sequence is used unless either the NOOLF or NONAMES option is specified:

1st priority Options from the command line

2nd priority Options from the user-specified Options List file

3rd priority Options from SUPERC OLIST A

4th priority Options from the LINE_DEF tag from the SUPERC NAMES file

Note: If you specify an option in the command line that *conflicts* with an option in the Options List file, the option in the Options List file takes precedence. (SuperC lists the conflicting option in the output listing.)

Compares From FILELIST

You can invoke SuperC from FILELIST. Specify the *new* and *old* files followed by a "(" and the SuperC options.

The *new* file can be selected from the list of files by using a "/" in the prefix area)

The *old* file can be named in full (fn ft fm) or, if appropriate, an "=" can be used to replicate that part of the *new* file identifier.

An example of SuperC being invoked from the FILELIST is shown in Figure 60. In this case, the two files are compared with:

- A compare type of LINE (the default)
- XEDIT being invoked to display the results of the comparison.
- A listing type of OVSUM
- No options to be used from an Options List file

.

```
 JLEVERI  FILELIST A0  V  108  Trunc=108 Size=657 Line=1 Col=1 Alt=9
Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      NEW3     TESTCASE C1 F         80         10          1  06/11/04 17.48.03
      NEW53    TESTCASE C1 V        125       2958         30  06/11/04 17.48.03
      NEW60    TESTCASE C1 V        100         64          1  06/11/04 17.48.03
      NEW59    TESTCASE C1 V         74         75          1  06/11/04 17.48.03
asmfsupc / old testcase c1 (xedit ovsum noolf   75          1  06/11/04 17.48.03
      NEW56    TESTCASE C1 V         71         22          1  06/11/04 17.48.03
      NEW13    TESTCASE C1 F         80         15          1  06/11/04 17.48.03

1= Help      2= Refresh  3= Quit    4=Sort(type)  5= Sort(date)  6= Sort(Size)
7= Backward  8= Forward  9= FL /n  10=           11= XEDIT      12= Cursor
```

Figure 60. Example of Invoking SuperC from FILELIST

Invoking the Comparison on VSE

On VSE, you invoke the SuperC Comparison as a batch program. You can use the SuperC Comparison on VSE to compare:

- Two sequential files (VSAM-managed or non-VSAM-managed)
- Two tape files
- One sequential file (VSAM-managed or non-VSAM-managed) and one tape file
- Two complete Librarian sublibraries
- Members of two Librarian sublibraries

The following examples describe the job control statements needed for each of these comparisons. Study the first example before looking at subsequent examples. Only the first example describes each statement in detail; subsequent examples describe only the statements specific to that comparison.

VSE JCL Example 1: Non-VSAM-managed Sequential Files

Figure 61 on page 200 shows simplified VSE JCL for comparing two non-VSAM-managed sequential files. This example is supplied with SuperC in the Librarian member ASMFVSC1.Z.

Before running this example, edit the lines highlighted by numbers (such as **1**) as described in the instructions following the example listing.

```
// JOB ASMFVSC1
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
/*                                                              ┐
/* Define "new" file                                           │
/*                                                             │
// DLBL new_file_name,'new_file_ID',0,SD                       │
// EXTENT extent_information                                   │
// ASSGN assign_logical_unit_information                       ├─ 1
/*                                                             │
/* Define "old" file                                           │
/*                                                             │
// DLBL old_file_name,'old_file_ID',0,SD                       │
// EXTENT extent_information                                   │
// ASSGN assign_logical_unit_information                       ┘
/*
/* Define update file (if required)
/*                                                             ┐
// DLBL update_file_name,'update_file_ID',0,SD                 │
// EXTENT extent_information                                   ├─ 2
// ASSGN assign_logical_unit_information                       ┘
/*
/* Note: The listing file is output to SYSLST
/*       (If the WIDE process option is used, SYSLST must be
/*       assigned to a printer capable of handling lines of
/*       at least 202 characters.)
/*
/* Run the compare with these options...
/*
// EXEC ASMFSUPC,PARM='options'                                 3
*
*  ...and these process statements
*
NEWDD new_file_name,attributes                                  ┐ 4
OLDDD old_file_name,attributes                                  ┘
UPDDD update_file_name                                          5
other_process_statements                                        6
  ⋮
/*
/&
```

Figure 61. Sample VSE JCL for Comparing Non-VSAM-Managed Sequential Files

1 Replace *new_file_name* and *old_file_name* with your choice of DLBL names for the files to be compared; also insert these DLBL names in the NEWDD and OLDDD process statements (see **4**). Replace *new_file_ID* and *old_file_ID* with the names of the files to be compared. Insert appropriate extent information and assign logical unit information.

Note: The terms “new” and “old” are used only for the sake of identifying the files being compared, which might or might not be different versions of the same file.

2 (Only needed if you specify a “UPD...” process option; see **3**.)

Replace *update_file_name* with your choice of DLBL name for the update file; also insert this DLBL name in the UPDDD process statement (see **5**). Replace *update_file_ID* with the name of the update file that you want SuperC to create. Insert appropriate extent information and assign logical unit information.

Most update files are intended to be used as input to other tools, rather than being “human-readable” reports (such as the listing file; see **3**). For instance, if you specify the UPDMVS8 process option, SuperC creates an

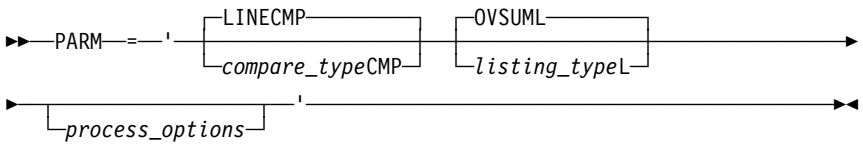
update file that you can use with VSE Librarian. You can use VSE Librarian to apply to the old file any updates that SuperC found in the new file.

SuperC creates the update file as a non-VSAM-managed sequential file. The file format, record size and block size depend on the "UPD..." process option you specified.

For more information on producing an update file, see the process options whose keywords start with "UPD" on Table 25 on page 226. For a selection of sample update files, see "Update Files" on page 292.

3 Replace `PARM='options'` with a PARM parameter in the following format:

Format of ASMFSUPC PARM parameter for SuperC Comparison



Note: Each option may be separated by either a space or a comma.

compare_type
The type of comparison you want performed: FILE, LINE, WORD, or BYTE. When specifying the compare type in the PARM parameter, add the suffix "CMP" (for example, WORD becomes WORDCMP).

For a description of each compare type, see "Compare Type" on page 186.

listing_type
The type of listing you want from the comparison: OVSUM, DELTA, CHNG, LONG, or NOLIST. When specifying the listing type in the PARM parameter, add the suffix "L" (for example, CHNG becomes CHNGL).

For a description of each listing type, see "Listing Type" on page 186.

process_options
Process options are keywords that direct SuperC how to perform the comparison or format the listing. Process options can be separated by spaces or commas.

For a description of each process option, see "Process Options" on page 226.

For example:

```
PARM='LINECMP DPCBCMT DELTAL NOSUMS'
```

instructs SuperC to:

- Perform a line-by-line comparison. (LINE compare type with "CMP" suffix.)
- Ignore COBOL comment lines. (Process option DPCBCMT ignores lines with an "*" in column 7.)
- Produce a listing showing changes, without an overall summary section. (Process option NOSUMS eliminates the group and final summary listing from the output listing.)

SuperC outputs the listing file to SYSLST. For a selection of sample listing files, see "Understanding the Listings" on page 269.

4 NEWDD and OLDDD are process statements that allow you to:

- Use your own choice of DLBL name for the new file and old file. If you do not specify NEWDD and OLDDD process statements, you must use the DLBL names NEWDD and OLDDD.
- Specify file attributes for the new and old files. If you do not specify NEWDD and OLDDD process statements with file attributes, SuperC assumes that the (non-VSAM) new and old files contain fixed-length unblocked records with a record size and block size of 80.

For more information on the NEWDD and OLDDD process statements, see "DD-VSE DLBL/TLBL Definitions" on page 246.

5 The UPDDD process statement allows you to use your own choice of DLBL name for the update file. If you do not specify an UPDDD process statement, you must use the DLBL name UPDDD for the update file.

6 Insert any other process statements (one per line) that you want to use.

For example, the following process statements:

```
CMPCOLM 7:72
LSTCOLM 1:72
```

instruct SuperC to compare only columns 7 to 72 in the new and old files (say, for comparing COBOL source without comparing sequence numbers), but to include in the listing file columns 1 to 72 (that is, the listing *contains* the sequence numbers).

For more information on process statements, see "Process Statements" on page 236.

VSE JCL Example 2: VSAM-managed Sequential Files

Figure 62 shows simplified VSE JCL for comparing two VSAM-managed sequential files. This example is supplied with SuperC in the Librarian member ASMFVSC2.Z.

```
// JOB ASMFVSC2
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// DLBL NEWDD,'new_file_ID',0,VSAM,DISP=(OLD,KEEP)
// DLBL OLDDD,'old_file_ID',0,VSAM,DISP=(OLD,KEEP)
// DLBL UPDDD,'update_file_ID',0,SD
// EXTENT extent_information
// ASSGN assign_logical_unit_information
// EXEC ASMFSUPC,PARM='options'
process_statements
  ⋮
/*
/&
```

Figure 62. Sample VSE JCL for Comparing VSAM-Managed Sequential Files

The update file, which is created by SuperC only if you specify an "UPD..." process option, is a non-VSAM-managed sequential file.

If you want to use DLBL names other than NEWDD, OLDDD, and UPDDD, then you must specify NEWDD, OLDDD, and UPDDD process statements (as shown in Figure 61 on page 200).

If you specify file attributes with the NEWDD and OLDDD process statements, then those file attributes override the VSAM catalog entries for the new and old files.

VSE JCL Example 3: VSAM Files

Figure 63 shows simplified VSE JCL for comparing two VSAM files. This example is supplied with SuperC in the Librarian member ASMFVSC3.Z.

```
// JOB ASMFVSC3
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// DLBL NEWDD,'new_file_ID',,VSAM,CAT=IJSYSUC
// DLBL OLDDD,'old_file_ID',,VSAM,CAT=IJSYSUC
// DLBL UPDDD,'update_file_ID',0,SD
// EXTENT extent_information
// ASSGN assign_logical_unit_information
// EXEC ASMFSUPC,PARM='options'
process_statements
  ⋮
/*
/&
```

Figure 63. Sample VSE JCL for Comparing VSAM Files

The update file, which is created by SuperC only if you specify an “UPD...” process option, is a non-VSAM-managed sequential file.

If you want to use DLBL names other than NEWDD, OLDDD, and UPDDD, then you must specify NEWDD, OLDDD, and UPDDD process statements (as shown in Figure 61 on page 200). In native VSAM, the file's attributes are taken from the VSAM catalog.

VSE JCL Example 4: Tape Files

Figure 64 shows simplified VSE JCL for comparing two labeled tape files. This example is supplied with SuperC in the Librarian member ASMFVSC4.Z.

Note: For *unlabeled* tape input, no `// TLBL` statement is used for the file concerned.

```
// JOB ASMFVSC4
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// TLBL new_file_name,'new_file_ID'                    1
// ASSGN SYS001,physical_unit_information              2
// TLBL old_file_name,'old_file_ID'                    3
// ASSGN SYS002,physical_unit_information              4
// DLBL update_file_name,'update_file_ID',0,SD
// EXTENT extent_information
// ASSGN assign_logical_unit_information
// EXEC ASMFSUPC,PARM='options'
NEWDD new_file_name,attributes                         5
OLDDD old_file_name,attributes
UPDDD update_file_name
other_process_statements
  ⋮
/*
/&
```

Figure 64. Sample VSE JCL for Comparing Labeled Tape Files

1 Replace *new_file_name* with your choice of TLBL name for the new file to be compared; also insert this TLBL name in the NEWDD process statement (see **5**). Replace *new_file_ID* with the name of the new file to be compared.

2 Insert appropriate physical unit information for the tape unit holding the new tape file.

3 Replace *old_file_name* with your choice of TLBL name for the old file to be compared; also insert this TLBL name in the OLDDD process statement (see **5**). Replace *old_file_ID* with the name of the old file to be compared.

4 Insert appropriate physical unit information for the tape unit holding the old tape file.

5 NEWDD and OLDDD are process statements that, for tape input, allow you to:

- Use your own choice of TLBL name for the new file and old file. If you do not specify NEWDD and OLDDD process statements, you must use the TLBL names NEWDD and OLDDD.
- Specify file attributes for the new and old files. If you do not specify NEWDD and OLDDD process statements with file attributes, SuperC assumes that the new and old files are fixed unblocked with a record size and block size of 80.

For more information on the NEWDD and OLDDD process statements, see “DD-VSE DLBL/TLBL Definitions” on page 246.

VSE JCL Example 5: Librarian Members

Figure 65 shows simplified VSE JCL for comparing all like-named members in two Librarian sublibraries. This example is supplied with SuperC in the Librarian member ASMFVSC5.Z.

```
// JOB ASMFVSC5
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// DLBL UPDDD,'update_file_ID',0,SD
// EXTENT extent_information
// ASSGN assign_logical_unit_information
// EXEC ASMFSUPC,PARM='options'
NEWDD newlib.sublib
OLDDD oldlib.sublib
other_process_statements
  ⋮
/*
/&
```

Figure 65. Sample VSE JCL for Comparing All Like-Named Members in Two Sublibraries

Members in either sublibrary not having like-named members in the other sublibrary are not compared, but are reported in the listing file.

The update file, which is created by SuperC only if you specify an “UPD...” process option, is a non-VSAM-managed sequential file. If you want to use a DLBL name other than UPDDD for the update file, specify an UPDDD process statement (as shown in Figure 61 on page 200).

To restrict a comparison of sublibraries to selected members only, use the SELECT process statement. For example, the following process statement:

```
SELECT NEW1.SOURCE:OLD1.SOURCE,SAME.C
```

instructs SuperC to compare only:

- Member NEW1.SOURCE in the new sublibrary with member OLD1.SOURCE in the old sublibrary
- Member SAME.C in the new sublibrary with member SAME.C in the old sublibrary

For more information on the SELECT process statement, see “Select Members (VSE)” on page 259.

To compare two Librarian members only, you can either:

- Use the SELECT process statement

or

- In the the NEWDD and OLDDD process statements, specify members rather than entire sublibraries. For example:

  ```
  NEWDD LIB.NEWSUB.SAME.C
  OLDDD LIB.OLDSUB.SAME.C
  ```

To compare groups of members, specify asterisk (*) wildcard character in the member name or type in the NEWDD and OLDDD process statements. For example:

```
NEWDD LIB.NEWSUB.NEW*.*
OLDDD LIB.OLDSUB.OLD*.*
```

Invoking the SuperC Search

The SuperC Search runs on MVS, CMS, and VSE. The following sections describe how to invoke the SuperC Search on each of these platforms.

Invoking the Search on MVS

On MVS, you invoke the SuperC Search as a batch program. You can use the SuperC Search on MVS to search:

- A sequential data set
- One, several, or all the members of a partitioned data set
- A VSAM data set
- A concatenated data set

MVS JCL Example

Figure 66 on page 206 shows simplified MVS JCL to run the SuperC Search. This example is supplied with SuperC in the sample PDS (default is ASM.JMQ415A.SASMSAM2) as member ASMFMVS1.

Before running this example, edit the lines highlighted by numbers (such as **1**) as described in the instructions following the example listing.

```
  ⋮
/*
/* Run the search with these options (see 1 and 6 )
/*
//RUN      EXEC PGM=ASMFSUPC,REGION=4M,PARM='SRCHCMP process_options'      1
/*
//STEPLIB  DD   DSN=#hlq.SASMMOD2,DISP=SHR                                 2
/*
/* Define data set to be searched
/*
//NEWDD    DD   DSN=search_file,DISP=SHR                                   3
/*
/* Direct listing data set to SYSOUT
/*
//OUTDD    DD   SYSOUT=*                                                   4
//SYSIN    DD   *
SRCHFOR 'search_string'                                                    5
  ⋮
other_process_statements                                                   6
  ⋮
/*
//
```

Figure 66. Sample MVS JCL to Run the SuperC Search

1 Replace *process_options* with any process options you want to use to customize how SuperC performs the search or formats the listing.

For a description of each process option, see “Process Options” on page 226.

For example:

```
PARM='SRCHCMP DPCBCMT COBOL'
```

instructs SuperC to perform a search:

- Ignoring COBOL comment lines. (Process option DPCBCMT ignores lines with an “*” in column 7.)
- Ignoring columns 1 to 6. (Process option COBOL ignores columns 1 to 6 which are assumed to be sequence numbers.)

2 Replace *#hlq* with the high level qualifier where SuperC is installed (default load library is ASM.JMQ415A.SASMMOD2).

3 Replace *search_file* with the item to be searched. This can be:

- A sequential data set
- A partitioned data set
- A member of a partitioned data set
- A VSAM data set
- A concatenated data set

If you specify a partitioned data set (PDS) name for *search file*, SuperC searches all members in the PDS.

To restrict the search of a PDS to selected members only, use the SELECT process statement. For example, the following process statement:

```
SELECT TEST1,TEST2
```

instructs SuperC to search only members TEST1 and TEST2 of the PDS specified by *search file*.

For more information on the SELECT process statement, see “Select PDS Members (MVS)” on page 260.

4 The listing data set, listing the results of the comparison. For example listings, see “Understanding the Listings” on page 269.

5 Replace *search_string* with a string that you want to search for. For information on specifying search strings, see “Search Strings in the Input File” on page 255.

6 Insert any other process statements (one per line) that you want to use.

For example, the following process statement:

```
DPLINE 'ignore this line'
```

instructs SuperC to exclude from the search any lines containing the specified string (“ignore this line”).

For more information on process statements, see “Process Statements” on page 236.

Invoking the Search on CMS using Menu Input

You can use the SuperC Search on CMS to search:

- A file
- A file group
- A member within a MACLIB or TXTLIB
- Complete MACLIBs or TXTLIBs

This section describes how to use the Primary Search Menu. This menu allows you to specify the file to be searched, and other SuperC options. For information on invoking the SuperC Search directly from the command line without using the menu, see “Invoking the Search on CMS using Command Line Input” on page 215.

To display the Primary Search Menu (see Figure 67 on page 208):

- Enter

  ```
  ASMFSUPC SRCH
  ```

  on the CMS command line

or

- Press the PF6 key from the Primary Comparison Menu (see “Invoking the Comparison on CMS using Menu Input” on page 183).

```
HLASM Toolkit Feature SuperC Compare Program - Search Menu
COMMAND ==>

                         Fn    Ft    Fm             (MACLIB/TXTLIB Files Only)
Search File ID     ==>                              Member  ==>

  Enter Search Strings and  Optional operands (WORD/PREFIX/SUFFIX and/or C)
           where C denotes Continuation/Additional Match String Requirement
 CAPS    ==>
 CAPS    ==>
 CAPS    ==>
 ASIS    ==>
 ASIS    ==>
 ASIS    ==>
                         Optional Section
Selection List     ==>                              ( NO / * )

Process Options    ==>
Listing File ID    ==>                              ( srchfn SRCHFOR A /file-id)

Process Stmts ID   ==>                              ( file-ID                  )
Auto Display Pgm   ==>                              ( BROWSE/XEDIT/EPDF etc.   )

 1=Help  3/6=Primary Menu   4=Quit   5=Proc Stmts   8=Proc Opts   ENTER/10=Exec
```

Figure 67. SuperC Primary Search Menu

Suppose you want to search for the uppercase string “ABCD” in the file NEW TEST1 A:

1. Enter the name of the file to be searched in the **Search File ID** field:

   ```
   Search File ID   ==> new test1 a
   ```

2. Enter the string to be searched for in one of the (three) **CAPS** fields:

   ```
   CAPS     ==> ABCD
   ```

3. Press Enter

The result of the search is then displayed.

Here are descriptions of each input field on the SuperC Primary Search Menu. Default values are underlined.

COMMAND

This field allows you to issue CP and CMS commands, such as FILELIST, ERASE, or RDRLIST.

Search File ID

This field specifies the name of the file to be searched. In almost all cases, this is a required field.[1]

SuperC allows the CMS convention of including wildcard characters (“*”) and equal signs (“=”) as part of the file ID. (However, only the “*” wildcard character applies for the Search File ID.)

The following examples show the effect of various entries in the Search File ID field:

[1] If a Process Statements file is specified and it contains any SELECTF process statements, the Search File ID name is ignored.

File ID Specified	Meaning
`new test1 a`	Single CMS file
`new test* a`	File group (all with filename “NEW” and a filetype starting with “TEST” and filemode “A”)
`new maclib`	The entire macro library, NEW

Note: A macro or text library with a filename containing an “*” (for example, ABC* MACLIB A or * TXTLIB C) is not processed as individual MACLIB/TXTLIBs with members. There is no method for specifying the “concatenation” of more than one MACLIB/TXTLIB.

Member This field specifies the name of the member, within either a macro library (MACLIB) or text library (TXTLIB), to be searched. (This field is only used when the file specified in Search File ID refers to a macro or text library.) If left blank, all members within the specified library are selected for the search.

File ID Specified	Member	Meaning
`new maclib c`	`xyz`	XYZ member in NEW MACLIB C.
`new maclib c`	`*`	All members in NEW MACLIB C. **Note:** The Selection List field (see page 211) must = NO.

Search String Fields (CAPS, ASIS)

You can specify up to 6 different strings to be searched for in a single search. Strings can be entered in any of the three CAPS entry fields and in any of the three ASIS entry fields. (The differences between the CAPS and ASIS entry fields are explained later in this section; see “Using the CAPS Entry Field” on page 210.)

You can specify strings as all uppercase characters, all lowercase characters, or a mixture of both. The case that you use depends on the entry field used (CAPS or ASIS).

A string may be further qualified as a word, prefix, or suffix, and where it appears on a line:

Qualifier	Meaning
WORD (or W)	String must appear as a separate *word*, that is, be delimited by spaces or special characters.
PREFIX (or P)	String must appear as the first part of some other text.
SUFFIX (or S)	String must appear as the last part of some other text.
C	Indicates continuation. The string must appear on the same line of input as the string defined in the previous entry line. (The two strings may appear in the input line in any order.) Strings without the “C” qualification are independent of previously specified lines. “C” may have further qualifiers:

+ The string in the "C" entry line must appear *after* the string specified in the previous entry line.

+n The string in the "C" entry line must start in the *n*th position after the string specified in the previous entry line.

column_range The string must start within this range of columns on a line.

Format is: *start_column*:*last_start_column*

Strings may be entered as a contiguous character string. If spaces are included in the string to be searched for, the entire search string must be enclosed within single quotes. If the string to be searched for contains single quotes, each embedded single quote must be represented by *two* single quotes in the search string.

When a string is qualified, the qualifier starts at the first non-space character after the (possibly quoted) string.

Hexadecimal strings must be specified using an "X" prefix followed by the hexadecimal string enclosed in single quotes. Such strings must contain an even number of valid hexadecimal characters (0 to 9, A to Z).

Using the CAPS Entry Field: Entering a string in one of the **CAPS** fields makes SuperC search for occurrences of the specified string in uppercase *only*. For example, if you enter the character string "abcd" in one of the CAPS fields, each occurrence of the string "ABCD" is searched for, but strings such as "aBcD" or "abcd" or "ABCd" are not sought.

The contents of each CAPS field is raised to uppercase after it is entered on the menu line.

Each of the following examples causes a search for all occurrences of the prefix "WXYZ":

```
CAPS          ==> WXYZ  prefix
CAPS          ==> wxyz  prefix
CAPS          ==> wXyZ  prefix
```

Note: If you use the Process Option ANYC (Any Case) in conjunction with the CAPS entry, then the string specified is searched for *regardless* of case.

Using the ASIS Entry Field: Entering a string in one of the ASIS fields makes SuperC search for occurrences of the specified string *exactly as specified*.

For example, if you enter the character string "abcd" in one of the ASIS fields, each occurrence of the string "abcd" is searched for, but strings such as "aBcD" or "ABCD" or "ABCd" are not sought.

The following example causes a search for all occurrences of the prefix "wXyZ":

```
ASIS          ==> wXyZ prefix
```

Examples of Search Strings:

Search String Specified	Searches For
`CAPS ==> ABC` `CAPS ==> efg`	Lines containing the string “ABC” or the string “EFG”
`CAPS ==> ABC WORD` `CAPS ==> EFG C`	Strings “ABC” and “EFG”on the same line; “ABC” must be a complete word
`ASIS ==> AbcD PREFIX`	All occurrences of the prefix “AbcD”
`CAPS ==> 'AB C''D'`	The string “AB C'D”
`CAPS ==> X'004CFF'`	The hexadecimal string X'004CFF'

`CAPS ==> ABC W 5:60`
`CAPS ==> EFG W C +`
`CAPS ==> HIJ C +5`
	The string “ABC” starting within columns 5 to 60 with the string “EFG” following somewhere in the same line and the string “HIJ” starting in the 5th position after “EFG”

Note: You can also specify search strings using any number of SRCHFOR and SRCHFORC process statements in a process statement file (see “Process Statements ID” on page 214). These search strings are used in addition to any search strings you specify in the menu.

Selection List

This field indicates if the Selection List facility is to be used. Valid values are:

NO Selection list facility not required.

***** Selection list facility required.

Note: The Selection List facility is only applicable when an “*” (asterisk) is contained within the Search File ID. (In the case of a macro or text file, an “*” must be contained within the specified Member name.)

Enter an “*” in the Selection List field to see a list of files from which to select the ones that you want.

The following examples illustrate the files that are listed for selection according to the file ID specified:

File ID Specified	Member	Files Listed for Selection
`new test1 *`		All files with the filename “NEW” and the filetype “TEST1”
`old test* a`		All files with the filename “OLD” and a filetype beginning with “TEST” and filemode “A”
`new txtlib a`	`*`	All members within the text library NEW TXTLIB A
`new maclib a`	`abc*`	All members within the macro library NEW MACLIB A whose name begins with “ABC”

For details about using the selection list, see “CMS File Selection List” on page 305 for using the selection list.

Process Options

You can specify the process options that you want (if any) by:

- Entering them directly in the process option line on the Primary Search Menu

or

- Selecting them from the Process Options Selection Menu (PF8)

For a full list and description of process options, see “Process Options” on page 226.

Entering Process Options Directly: If you choose to enter the process options directly, simply type in each process option keyword on the entry line (each keyword must be separated by a space). Up to 51 characters (including spaces) can be entered.

```
HLASM Toolkit Feature SuperC Compare Program - Search Menu
COMMAND ==>

                         Fn    Ft    Fm             (MACLIB/TXTLIB Files Only)

  ⋮

Process Options    ==>  anyc nosums


  ⋮
 1=Help  3/6=Primary Menu   4=Quit   5=Proc Stmts   8=Proc Opts   ENTER/1
```

Figure 68. SuperC Primary Search Menu with Process Options Entered Directly

Figure 68 shows two process options entered directly on the Process Options line:

ANYC (“Any Case”)
NOSUMS (“No Summary Section”)

Selecting Process Options from the Menu: The second way to specify process options is to select them from the Process Options Selection Menu. This menu shows the process options that are valid for a search. To display this menu, press PF8 (see Figure 69 on page 213).

```
 HLASM Toolkit Feature SuperC Compare Program - Search Options          1 of 1)
 Search Options -------- SuperC Compare Program  ------------------ (1 of 1)
 COMMAND ==>
 Select option(s) from the following list or "blank" to remove.
/*GE Select option(s) from the following list or "blank" to remove.

 Sel                             Search Process Options
       SEQ     - Ignore sequence columns 73-80 on F 80 input source files,  or
       NOSEQ   - Process columns 73-80 as data on F 80 input source files,  or
       COBOL   - Ignore sequence columns 1-6 on F 80 input source files.

  S    ANYC    - Process text lines as upper case.
       IDPFX   - List filename/member as prefix to each search line found.
       XREF    - Cross references lines found for each search string.
       LPSF    - List search and up to six preceding and following lines, or
       LMTO    - List group totals only,                                    or
       LNFMTO  - List members/files where no lines were found,              or
       LTO     - List total summary only.

       LONGLN   - Lists up to 176 columns.  Maximum line length = 202/203.
       NOPRTCC  - No print control column and page separators.
       APNDLST  - Append listing report to listing data set.
  S    NOSUMS   - Generate no summary section in the report listing.
  Others:DPACMT,DPADCMT,DPBLKCL,DPCBCMT,DPFTCMT,DPPDCMT,DPPLCMT,DPPSCMT,DPMACMT
      (Enter these keywords directly on the main menu options selection lines)
  PF1=Help                        PF3=Menu                          PF8=Menu
```

Figure 69. Example of the SuperC Process Options Selection Menu (Search)

To select a process option, enter an “S” next to it.

Process options which have been selected previously appear with an “S” alongside them (as for ANYC and NOSUMS in Figure 69).

When you no longer require a process option, you can either clear the “S” from the Process Options Selection Menu or delete the option keyword from the Process Option field on the Primary Search Menu.

Listing File ID

This field specifies the name of the listing file generated as a result of the search. (The SuperC Search always generates a listing file.)

You can:

- Leave this field blank (in which case SuperC allocates a default name for the listing file)
- Specify a full file ID to be used for the listing file
- Use “*” and “=” symbols (which results in the listing file ID being a combination of the *fn ft fm* specified in the Search File ID and the details you enter for the Listing File ID)

This is best illustrated by some examples:

Search File ID	Listing File ID	File ID Used
new test a		new srchfor a
new test a	myname mytype a	myname mytype a
* test a		$ srchfor a
new test a	= listing a	new listing a
new* test a	* listing a	new$ listing a

Process Statements ID

This field specifies the name of the file (if any) containing the process statements to be used in the search.

Process statements (which are similar to process options but require one or more additional items of information to be specified) are always passed to SuperC in a file.

For a full list and description of process statements, see “Process Statements” on page 236.

You can either enter the name of an existing file that contains process statements, or press PF5 to create a new file and specify the process statements.

Pressing PF5 displays the Process Statements Entry Menu (see Figure 70) showing examples of some of the process statements available and allows you to enter the process statements that you want.

When you exit from the Primary Search Menu, SuperC automatically generates a file (called SRCHFOR SYSIN A) containing each of the process statements you have specified. (`SRCHFOR SYSIN A` is entered against “Process Stmts ID” on the Primary Search Menu.)

```
HLASM Toolkit Feature SuperC Compare Program - Search Statements       (1 of 1)


Enter Process Statements for Statements File:
   ==>

     Examples                              Explanation

  SRCHFOR  'ABCD' W           Search for the word "ABCD"
  SRCHFORC 'DEFG'             "DEFG" must be on same line as word "ABCD"
  CMPCOLM  1:60  75:90        Search columns 1:60 and 75:90 for string(s).

 ⋮


 PF1=Help       PF3=Menu       PF5=Menu       PF6=Cancel       ENTER=Save Line
```

Figure 70. Example of the SuperC Process Statements Entry Menu (Search)

Note: When you press PF5, SuperC erases any existing SRCHFOR SYSIN A file before creating the new file.

Auto Display Pgm

This field specifies the name of an editor or browse program to inspect the search results:

program name The name of a valid editor or browse program to be invoked to display the results of the search. (For example, *XEDIT*, *EPDF*, *BROWSE*)

(space) Results of search not displayed. Editor defaults to XEDIT.

Note: If no strings are found in the search:

1. The search results are not displayed.
2. The output listing file is still generated.

Primary Search Menu PF Key Definitions

PF1 Help. Displays the first search help menu.

PF3 SuperC exits from the search process and goes (or returns) to the Primary Comparison Menu.

PF4 Quit. Terminates the search. Returns to the environment before SuperC.

PF5 Proc Stmts (Process Statements). Displays the Process Statements Entry Menu. This menu contains some examples of the more widely-used process statements. It also has a field to allow you to input one process statement at a time into the SRCHFOR SYSIN A file.

Note: When you press PF5, SuperC erases any existing SRCHFOR SYSIN A file before creating the new file.

PF8 Proc Opts (Process Options). Displays the Process Options Selection Menu.

PF10 Execute and Quit. Verifies user-input fields, invokes SuperC, and returns to the environment before SuperC.

ENTER Execute. Verifies user-input fields and invokes SuperC. After the search has completed, control returns to the environment before the SuperC Search was invoked.

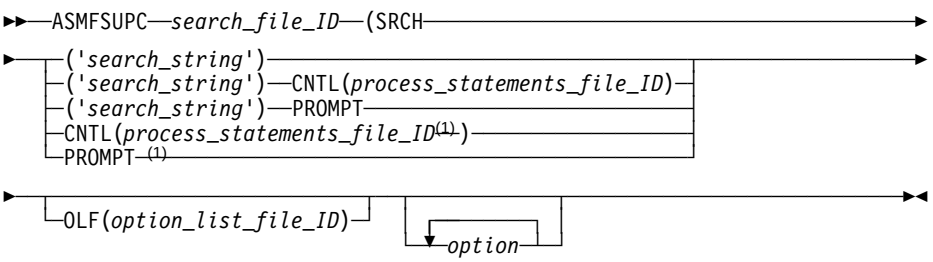
Invoking the Search on CMS using Command Line Input

You can use the SuperC Search on CMS to search:

- A file
- Selected files within a file group
- A file group
- A member within a MACLIB or TXTLIB
- Complete MACLIBs or TXTLIBs

This section describes how to invoke the SuperC Search directly from the command line, without using the Primary Search Menu. For information on using this menu, see “Invoking the Search on CMS using Menu Input” on page 207.

To invoke the SuperC Search from the CMS command line, use the following format:



Note:

1 When no search string is entered on the ASMFSUPC command line, it must be specified by a SRCHFOR process statement–either contained in a process statements file specified by the keyword CNTL, or entered via the Process Statements Entry Menu (displayed using the keyword PROMPT).

search_file_ID The name of the file (or library) to be searched.

SRCH If followed by a search string (within parentheses and single quotes), specifies the string to be used in the search.

Note: When the search string is entered directly in the command line in this way, it is searched for in uppercase only. (If you want the string searched for regardless of case, use the ANYC process option.)

If *not* followed by a search string, one of the following must be used:

1. The keyword CNTL to specify the process statements file containing one or more SRCHFOR process statements, each specifying a search string.
2. The keyword PROMPT to display the Process Statements Entry Menu from which you can enter one or more SRCHFOR process statements (each specifying a search string). SuperC automatically enters these statements in the process statements file SRCHFOR SYSIN A.

Notes:

1. It is possible to specify a search string in the command line *and* one or more additional search_strings (using SRCHFOR process statements) in the process statements file:
   - Whose ID follows the keyword CNTL, or
   - SRCHFOR SYSIN A (using the keyword PROMPT to display the Process Statements Entry Menu).
2. The keywords CNTL and PROMPT cannot be used together. For further details, see “Process Statement ID” on page 219.

OLF This is the keyword for the Options List File (when used). For further details, refer to “Options List File” on page 220.

option Each type of option is described in detail following “Types of Options” on page 217.

Also, see “CMS Command Line Option Directives” on page 266

and “CMS Command Line Statement Option Directives” on page 267.

Examples of Invoking the SuperC Search on the CMS Command Line:

To search for the string “ABC” in file TEST1 NEW A, enter:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC')
```

To search for the string “ABC” in file TEST1 NEW A, using a SRCHFOR process statement in your own-named process statements file MYPROC FILE A:

- Specify the process statement `SRCHFOR 'ABC'` in the file MYPROC FILE A
- Enter:

  ```
  ASMFSUPC TEST1 NEW A (SRCH CNTL(MYPROC FILE A)
  ```

To search for the string “ABC” in file TEST1 NEW A, using a SRCHFOR process statement entered via the Process Statements Entry Menu:

- Enter:

  ```
  ASMFSUPC TEST1 NEW A (SRCH PROMPT
  ```

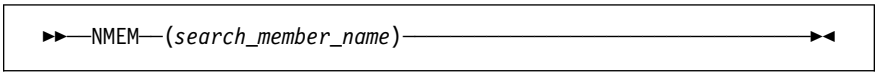
- When the Process Statements Entry Menu is displayed, enter `SRCHFOR 'ABC'`

Types of Options

You can specify any of the following options in the CMS command line or in the Options List file (see page 220):

Member Names

This option specifies the name of a member within a library.



For example:

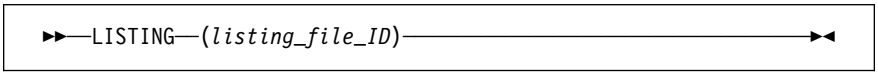
```
ASMFSUPC MACLIB NEW A (SRCH('DEF') NMEM(MEMB1)
```

searches for the string “DEF” in member MEMB1 in the macro library NEW.

Note: Member names can only be used as options when the *search_file_ID* specified refers to either a macro or text library.

Listing File

This option specifies the alternative name to be assigned to the listing file generated as a result of the search process.



For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') LISTING(CMPLIST RESULTS A)
```

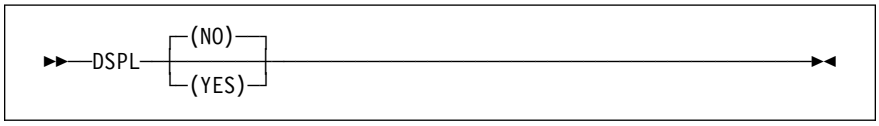
creates a listing file named CMPLIST RESULTS A.

Note: If you do not use the LISTING option, the listing file is generated with a default ID consisting of:

fn Filename of the search file
ft SRCHFOR
fm A

Display Output

This option determines if the results of the search are to be displayed.



NO
Do not display output

YES
Display output

For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') DSPL(YES)
```

causes the search results to be displayed.

Note: If you specify an editor or browse program (see following option), the "Display Output" option defaults to YES.

For a further description of the Display Output option, see "Display Output" on page 190.

Auto Display Program

This option is used in conjunction with the Display Output option. It allows you to use the editor or browse program of your choice (provided that it is supported in your processing environment). The default is XEDIT.

Examples:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') EPDF

ASMFSUPC TEST1 NEW A (SRCH('ABC') XEDIT

ASMFSUPC TEST1 NEW A (SRCH('ABC') BROWSE
```

The above examples specify editors EPDF and XEDIT, and browse program BROWSE, respectively.

For a further description of the Auto Display Program, see page 214.

Process Options

This option specifies the process options to be used in the search process.

These can be one or more of the process option keywords which are valid for the SuperC Search. For details of these, see "Process Options" on page 226.

For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') ANYC
```

specifies the process option ANYC so that the string “ABC” is searched for regardless of case. Matches are found, for example, with “abc” and “AbC” and “ABc” in file TEST1 NEW A.

Option Directives

You can use any of the following option directive keywords:

ERASRC0 Erase listing file if no differences
NOIMSG No information messages
NOOLF No Options List file
PRINT Print results

For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') NOOLF
```

specifies that any options contained in the Options List file are not to be used in the search.

For further descriptions of each Option Directive, see “CMS Command Line Option Directives” on page 266.

Process Statement Directives

The following directives are transformed into process statements. They can be *one* of the following keywords:

CC Compare columns
LC List columns
LT Line count

For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') LC(7:14)
```

selects columns 7 to 14 to be listed in the output.

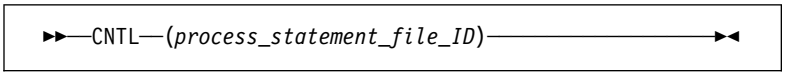
For further descriptions of each process statement directive, see “CMS Command Line Statement Option Directives” on page 267.

Process Statement ID

This option specifies how process statements are to be supplied to the SuperC Search.

One of two keywords can be used:

CNTL Use the CNTL keyword if you want to use an existing file that contains the process statements you require.



For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') CNTL(TSTPRO OPTS A)
```

specifies that the process statements in file TSTPRO OPTS A are to be used.

PROMPT

PROMPT indicates to SuperC that the file SRCHFOR SYSIN A is to be used to supply the process statements and causes the Process Statements Entry Menu to be displayed. This menu contains examples of the more widely-used process statements.

It also has a field that allows you to input one process statement at a time into the SRCHFOR SYSIN A file.

For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') PROMPT
```

causes the Process Statements Entry Menu to be displayed.

Note: If SRCHFOR SYSIN A already exists, its contents are erased before creating the new file.

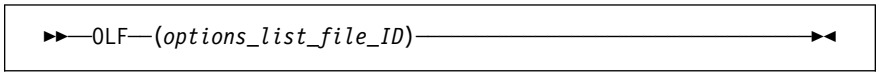
Options List File

You can use the Options List file to hold a set of default options (to save you entering them each time on the CMS command line). Any of the options described in this section can be placed in the Option List file and take effect unless overridden by options in the command line.

If you do not specify a name for the Options List file, SuperC looks for a file with the default name SRCHFOR OLIST A and, if found, uses the options contained in that file for the search process.

However, you can nominate an alternative Options List file by using the keyword OLF. OLF allows you to specify either a fully-qualified file ID (*fn ft fm*) or a partially-qualified file ID for the Options List file that you want SuperC to use (see “Default Naming Convention for Options List file” on page 220).

Note: SuperC uses options contained in an OLF-specified Options List file before those in SRCHFOR OLIST A (see “Command Line Priority and Overriding” on page 221).



For example:

```
ASMFSUPC TEST1 NEW A (SRCH('ABC') OLF(MYOPTS FILE A)
```

specifies that the options in file MYOPTS FILE A are to be used.

Note: Not all options in the Options List file can be overridden since there is no way to negate them. Take care when considering which options to include in the file when using OLF.

To examine this further, let's look at an example of an Options List file containing the following:

```
XEDIT CNTL(SRCHFOR STMTS A)
```

If the Options List file that you nominate in the CMS command line (by using the OLF keyword) contains the above options, you cannot override the Process Statements ID keyword CNTL (and therefore the process statements contained in the file SRCHFOR STMTS A take effect).

Default Naming Convention for Options List file: The command line uses the following defaults in the naming of the Options List file:

Command Line	OLF ID Used
`ASMFSUPC...(...`	SRCHFOR OLIST A
`ASMFSUPC...(NOOLF...`	(none)
`ASMFSUPC...(OLF(TST1)...`	TST1 OLIST A
`ASMFSUPC...(OLF(TST1 OPTS)...`	TST1 OPTS A
`ASMFSUPC...(OLF(TST1 OPTS A)...`	TST1 OPTS A

Command Line Priority and Overriding

The following priority sequence is used unless the NOOLF option is specified:

1st priority Options from the command line

2nd priority Options from the user-specified Options List file

3rd priority Options from SRCHFOR OLIST A

Note: If you specify an option in the command line that *conflicts* with an option in the Options List file, the option in the Options List file takes precedence. (SuperC lists the conflicting option in the output listing.)

SRCH Process Statement Directive

SRCH is is the command line directive for the SRCHFOR process statement and is used only for the SuperC Search.

SRCH specifies:

- The string that is to be searched for
- An optional string *qualifier*:

  W Word. String must appear as a separate *word*. That is, be delimited by one or more spaces or special characters.

  P Prefix. String must appear as the *first* part of some other text.

  S Suffix. String must appear as the *last* part of some other text.
- A *relational operator* (where there is more than one string specified):

  & *And*. Both of the strings (either side of the “&”) must appear on the same line.

  | *Or*. At least one of the strings (either side of the “|”) must appear on the line.

  Up to 6 relational operators may be used in one SRCH. When more than one relational operator is used, they are processed from left to right. Parentheses are not permitted.

Example	Description
`ASMFSUPC...(SRCH('ABC' | 'DEF')`	Find a line with string “ABC” or string “DEF”
`ASMFSUPC...(SRCH('AB C' & 'DEF' W)`	Find a line with string “AB C” and word “DEF”

Invoking the Search on VSE

On VSE, you invoke the SuperC Search as a batch program. You can use the SuperC Search on VSE to search:

- A sequential file (VSAM-managed or non-VSAM-managed)
- A tape file
- One, several, or all of the members of a Librarian sublibrary

The following examples describe the job control statements required for each of these searches. Study the first example before looking at subsequent examples. Only the first example describes each required statement in detail; subsequent examples describe only the statements specific to that search.

VSE JCL Example 1: Non-VSAM-managed Sequential Files

Figure 71 shows simplified VSE JCL for searching a non-VSAM-managed sequential file. This example is supplied with SuperC in the Librarian member ASMFVSS1.Z.

Before running this example, edit the lines highlighted by numbers (such as **1**) as described in the instructions following the example listing.

```
// JOB ASMFVSS1
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
/*
/* Define file to be searched
/*
// DLBL search_file_name,'search_file_ID',0,SD                    ┐
// EXTENT extent_information                                      ├ 1
// ASSGN assign_logical_unit_information                          ┘
/*
/* Note: The listing file is output to SYSLST.
/*       (If the WIDE process option is used, SYSLST must be
/*       assigned to a printer capable of handling lines of
/*       at least 202 characters.)
/*
/* Run the search with these options...
/*
// EXEC ASMFSUPC,PARM='SRCHCMP process_options'                    2
*
*  ...and these process statements
*
NEWDD search_file_name,attributes                                 3
SRCHFOR 'search_string'                                           4
  ⋮
other_process_statements                                          5
  ⋮
/*
/&
```

Figure 71. Sample VSE JCL for Searching a Non-VSAM-Managed Sequential File

1 Replace *search_file_name* with your choice of DLBL name for the file to be searched; also insert this DLBL name in the NEWDD process statement (see **3**). Replace *search_file_ID* with the name of the file to be searched. Insert appropriate extent information and assign logical unit information.

2 Replace *process_options* with any process options you want to use to customize how SuperC performs the search or formats the listing.

For a description of each process option, see “Process Options” on page 226.

For example:

```
PARM='SRCHCMP DPCBCMT COBOL'
```

instructs SuperC to perform a search:

- Ignoring COBOL comment lines. (Process option DPCBCMT ignores lines with an “*” in column 7.)

- Ignoring columns 1 to 6. (Process option COBOL ignores columns 1 to 6 which are assumed to be sequence numbers.)

SuperC outputs the listing file to SYSLST. For a selection of sample listing files, see “Understanding the Listings” on page 269.

3 NEWDD is a process statement that allows you to:

- Use your own choice of DLBL name for the file to be searched. If you do not specify a NEWDD process statement, you must use the DLBL name NEWDD.
- Specify file attributes for the file to be searched. If you do not specify a NEWDD process statement with file attributes, SuperC assumes that the (non-VSAM) file to be searched contains fixed-length unblocked records with a record size and block size of 80.

For more information on the NEWDD process statement, see “DD-VSE DLBL/TLBL Definitions” on page 246.

4 Replace *search_string* with a string that you want to search for. For information on specifying search strings, see “Search Strings in the Input File” on page 255.

5 Insert any other process statements (one per line) that you want to use.

For example, the following process statement:

```
DPLINE 'ignore this line'
```

instructs SuperC to exclude from the search any lines containing the specified string (“ignore this line”).

For more information on process statements, see “Process Statements” on page 236.

VSE JCL Example 2: VSAM-managed Sequential Files

Figure 72 shows simplified VSE JCL for searching a VSAM-managed sequential file. This example is supplied with SuperC in the Librarian member ASMFVSS2.Z.

```
// JOB ASMFVSS2
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// DLBL NEWDD,'search_file_ID',0,VSAM,CAT=IJSYSUC,DISP=(OLD,KEEP)
// EXEC ASMFSUPC,PARM='SRCHCMP process_options'
SRCHFOR 'search_string'
  ⋮
other_process_statements
  ⋮
/*
/&
```

Figure 72. Sample VSE JCL for Searching a VSAM-Managed Sequential File

If you want to use a DLBL name other than NEWDD, then you must specify a NEWDD process statement (as shown in Figure 71 on page 222).

If you specify file attributes with the NEWDD process statement, then those file attributes override the VSAM catalog entries for the file to be searched.

VSE JCL Example 3: VSAM Files

Figure 73 shows simplified VSE JCL for searching a VSAM file. This example is supplied with SuperC in the Librarian member ASMFVSS3.Z.

```
// JOB ASMFVSS3
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// DLBL NEWDD,'search_file_ID',,VSAM,CAT=IJSYSUC
// EXEC ASMFSUPC,PARM='SRCHCMP process_options'
SRCHFOR 'search_string'
  ⋮
other_process_statements
  ⋮
/*
/&
```

Figure 73. Sample VSE JCL for Searching a VSAM File

If you want to use a DLBL name other than NEWDD, then you must specify a NEWDD process statement (as shown in Figure 71 on page 222). In native VSAM, the file's attributes are taken from the VSAM catalog.

VSE JCL Example 4: Tape File

Figure 74 shows simplified VSE JCL for searching a labeled tape file. This example is supplied with SuperC in the Librarian member ASMFVSS4.Z.

Note: For *unlabeled* tape input, no `// TLBL` statement is used.

```
// JOB ASMFVSS4
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// TLBL search_file_name,'search_file_ID'                    1
// ASSGN SYS001,physical_unit_information                   2
// EXEC ASMFSUPC,PARM='SRCHCMP process_options'
NEWDD search_file_name,attributes                           3
SRCHFOR 'search_string'
  ⋮
other_process_statements
  ⋮
/*
/&
```

Figure 74. Sample VSE JCL for Searching a Labeled Tape File

1 Replace *search_file_name* with your choice of TLBL name for the file to be searched; also insert this TLBL name in the NEWDD process statement (see **3**). Replace *search_file_ID* with the name of the file to be searched.

2 Insert appropriate physical unit information for the tape unit holding the tape file to be searched.

3 NEWDD is a process statements that, for tape input, allows you to:

- Use your own choice of TLBL name for the search file. If you do not specify a NEWDD process statement, you must use the TLBL name NEWDD.

- Specify file attributes for the search file. If you do not specify a NEWDD process statement with file attributes, SuperC assumes that the search file is fixed unblocked with a record size and block size of 80.

For more information on the NEWDD process statement, see “DD-VSE DLBL/TLBL Definitions” on page 246.

VSE JCL Example 5: Librarian Members

Figure 75 shows simplified VSE JCL for searching all members in a Librarian sublibrary. This example is supplied with SuperC in the Librarian member ASMFVSS5.Z.

```
// JOB ASMFVSS5
// LIBDEF *,SEARCH=(PRD2.PROD)
// OPTION NODUMP
// EXEC ASMFSUPC,PARM='SRCHCMP process_options'
NEWDD srchlib.sublib
SRCHFOR 'search_string'
  ⋮
other_process_statements
  ⋮
/*
/&
```

Figure 75. Sample VSE JCL for Searching All Members in a Sublibrary

To restrict the search to selected members only, use the SELECT process statement. For example, the following process statement:

```
SELECT TEST1.C,TEST2.C
```

instructs SuperC to search only members TEST1.C and TEST2.C in the sublibrary *srchlib.sublib*.

For more information on the SELECT process statement, see “Select Members (VSE)” on page 259.

To search only one Librarian member, you can either:

- Use the SELECT process statement

or

- In the NEWDD process statement, specify a member rather than a sublibrary. For example:

  ```
  NEWDD LIB.SRCHLIB.TEST1.C
  ```

To search a group of members in a sublibrary, specify asterisk (*) wildcard characters in the member name or type in the NEWDD process statement. For example:

```
NEWDD LIB.SRCHLIB.TEST*.*
```

Process Options

You can tailor the comparison or search using *process options* and *process statements*. Process options are single keywords, whereas process statements consist of a keyword and one or more operands. For details on process statements, see “Process Statements” on page 236.

On **MVS**, you specify process options in a JCL `PARM` parameter:

- For comparison processing, refer to “Invoking the Comparison on MVS” on page 180.
- For search processing, refer to “Invoking the Search on MVS” on page 205.

On **CMS**, you specify process options:

- On a menu
  - For comparison processing, refer to “Process Options” on page 188.
  - For search processing, refer to page 212.
- On the CMS command line
  - For comparison processing, refer to page 195.
  - For search processing, refer to page 218.
- By means of the Options List file
  - For comparison processing, refer to “Options List File” on page 197.
  - For search processing, refer to “Options List File” on page 220.

On **VSE**, you specify process options in a JCL `PARM` parameter:

- For comparison processing, refer to “Invoking the Comparison on VSE” on page 199.
- For search processing, refer to “Invoking the Search on VSE” on page 221.

Table 25 lists each of the process option keywords and shows the compare type for which each process option can be used. The table also shows if the process option is valid for the SuperC Search.

Following Table 25 are descriptions for each type of process option.

Table 25 (Page 1 of 3). Summary of Process Options

Process Option		Valid For				
		Compare Type				Search
Keyword	**Description**	**FILE**	**LINE**	**WORD**	**BYTE**	
ALLMEMS	All members	√	√	√	√	√
ANYC	Any case		√	√		√
APNDLST[1]	Append listing output	√	√	√	√	√
APNDUPD[1]	Append update		√	√	√	√
CKPACKL[1]	Check for packed format		√	√		
CNPML[2]	Count non-paired member/file lines		√			
COBOL[3]	For COBOL source files		√	√		√
COVSUM	Conditional summary	√	√	√	√	
DLMDUP	Do not list matching duplicate lines		√			
DLREFM	Do not list reformatted lines		√			
DPACMT	Do not process asterisk (*) comment lines		√	√		√

Table 25 (Page 2 of 3). Summary of Process Options

Process Option		Valid For				
		Compare Type				Search
Keyword	**Description**	**FILE**	**LINE**	**WORD**	**BYTE**	
DPADCMT	Do not process ADA-type comments		√	√		√
DPBLKCL	Do not process blank comparison lines		√	√		√
DPCBCMT	Do not process COBOL-type comment lines		√	√		√
DPCPCMT	Do not process C++ -type comment lines		√	√		√
DPFTCMT	Do not process FORTRAN-type comment lines		√	√		√
DPMACMT	Do not process PC Assembly-type comment lines		√	√		√
DPPLCMT	Do not process PL/I-type comments		√	√		√
DPPSCMT	Do not process Pascal-type comments		√	√		√
FMSTOP	Stop immediately a difference found	√				
FMVLNS	Flag moved lines		√			
GWCBL	Generate WORD/LINE comparison change bar listing		√	√		
IDPFX	Identifier-prefixed listing lines					√
LMCSFC [4,9]	Load module CSECT file compare	√				
LMTO[5]	List group member totals					√
LNFMTO[5]	List not-found member totals only					√
LOCS	List only changed entries in summary	√	√	√	√	
LONGLN[6]	Long lines		√			√
LPSF[5]	List previous-search-following lines					√
LTO[5]	List totals only					√
MIXED	Mixed input (single/double byte) text		√	√		
NARROW[6]	Narrow (side-by-side) listing		√			
NOPRTCC	No printer control columns	√	√	√	√	√
NOSEQ[3]	No sequence numbers		√	√		√
NOSUMS	No summary section		√	√	√	√
REFMOVR	Reformat override		√			
SDUPM[9]	Search duplicate members					√
SEQ[3]	Ignore standard sequence number columns		√	√		√
SYSIN[9]	Provide alternative DD name for process statements.	√	√	√	√	√
UPDCMS8[7]	Update CMS8 format		√			
UPDCNTL[7]	Update control		√	√	√	
UPDLDEL[7]	Update long control		√			
UPDMVS8[7]	Update MVS8 format		√			
UPDPDEL[7]	Update prefixed delta lines		√			
UPDREV[7]	Update revision		√	√		
UPDREV2[7]	Update revision (2)		√	√		
UPDSEQ0[7]	Update sequence 0		√			
UPDSUMO[7]	Update summary only		√	√	√	
WIDE[6]	Wide (side-by-side) listing		√			
XREF	Cross reference strings					√
XWDCMP	Extended word comparison			√		

Table 25 (Page 3 of 3). Summary of Process Options

Process Option		Valid For				
		Compare Type				Search
Keyword	Description	FILE	LINE	WORD	BYTE	
Y2DTONLY[8]	Compare Dates Only		√			

Note:

1. Not supported on VSE
2. Valid for group LINE comparisons only.
3. COBOL, SEQ, and NOSEQ are mutually exclusive.
4. Not supported for PDSE.
5. LMTO, LNFMTO, LPSF, and LTO are mutually exclusive.
6. LONGLN, NARROW, and WIDE are mutually exclusive.
7. All update (UPD) process options are mutually exclusive. Also, they cannot be used with the process option Y2DTONLY.
8. Y2DTONLY is not supported for change bar listing (process option GWCBL).
9. Supported on MVS only.

The following sections describe each process option in detail:

ALLMEMS Process all members in a PDS including ALIAS members. Without this process option, when performing a PDS compare, SuperC does not include members with the ALIAS attribute unless explicitly specified by a SELECT process statement. The ALLMEMS process option indicates that all directory entries including those with the ALIAS attribute are to be processed.

ANYC Any case. Lowercase alphabetic characters (a to z) in source files are translated to uppercase (A to Z) before comparison processing. (The actual input files are not modified.)

Use this option to cause strings such as “ABC,” “Abc,” “ABc,” to compare equally.

Valid for LINE and WORD compare types and Search.

APNDLST The APNDLST process option appends the listing output to the specified or default listing file. If the file does not exist, it is created.

APNDLST allows you to collect updates from multiple comparisons into one listing file.

Valid for FILE, LINE, WORD, and BYTE compare types and Search.

Notes:

1. You can also do this by using the SELECT process statement (and, on CMS, SELECTF) that identifies different files/members and produces a single listing.
2. APNDLST is not supported on VSE.

APNDUPD The APNDUPD process option appends the update output to the specified or default update file. If the file does not exist, it is created.

APNDUPD allows you to collect updates from multiple comparisons into one update file.

Valid for LINE, WORD, and BYTE compare types and Search.

Notes:

1. You can also do this by using the SELECT process statement (and, on CMS, SELECTF) that identifies different files/members and produces a single listing.
2. APNDUPD is not supported on VSE.

CKPACKL Check for packed format. This option determines if the member or sequential data set has the standard ISPF/PDF packed header format. If required, SuperC unpacks the input data set or member during the comparison.

Valid for LINE and WORD compare types.

Note: CKPACKL is not supported on VSE.

CNPML Count non-paired member/file lines for the group summary list. Use this option to inventory the total number of processed and not-processed lines. Otherwise, only the paired entries are listed with line counts.

Valid for LINE compare type.

Note: CNPML is only used when comparing a *group* of files.

COBOL Ignore columns 1 to 6 in both COBOL source files. Data in columns 1 to 6 is assumed to be sequence numbers.

Valid for LINE and WORD compare types and Search.

COVSUM Conditional summary section. List the final summary section and/or the update file for the option UPDSUMO only if there are differences. This is useful when used in combination with APNDLST or APNDUPD.

Valid for FILE, LINE, WORD, and BYTE compare types.

DLMDUP Do not list matching duplicate lines. *Old* file source lines that match *new* file source lines are omitted from the side-by-side output listing.

Valid for LINE compare type.

DLREFM Do not list reformatted lines. *Old* file source lines that have the same data content (that is, all data is the same except the position and number of space characters) as the *new* file lines are omitted from the listing. Only the *new* file reformatted lines are included in the output.

Valid for LINE compare type.

DPACMT Do not process asterisk (*) comment lines. Lines with an "*" in column 1 are excluded from the comparison set. Other forms of assembler comments are unaffected.

Valid for LINE and WORD compare types and Search.

DPADCMT Do not process ADA type comments. ADA comments are whole or partial lines that appear after the special "--" sequence. Blank lines are also considered part of the comment set. This option produces a comparison listing with comments removed and part comments blanked.

Valid for LINE and WORD compare types and Search.

DPBLKCL Do not process blank comparison lines. Source lines in which all of the comparison columns are blank are excluded from the comparison set.

Note: It is redundant to use this option with DPADCMT, DPPLCMT, or DPPSCMT as these process options also bypass blank comparison lines.

Valid for LINE and WORD compare types and Search.

DPCBCMT Do not process COBOL-type comment lines. COBOL source lines with an "*" in column 7 are excluded from the comparison set

Valid for LINE and WORD compare types and Search.

DPCPCMT Do not process C++ end-of-line type compiler comments. These are "//" delimited comments. DPPLCMT may also be used with DPCPCMT when the source file contains "/* ... */" comments delimiters.

Valid for LINE and WORD compare types and Search.

DPFTCMT Do not process FORTRAN-type comment lines. FORTRAN source lines with a "C" in column 1 are excluded from the comparison set.

Valid for LINE and WORD compare types and Search.

DPMACMT Do not process PC Assembly-type comments. This uses the IBM PC definition for assembler comments: comments begin with either the COMMENT assembler directive or a semi-colon (;).

Valid for LINE and WORD compare types and Search.

DPPLCMT Do not process PL/I-type comments. PL/I, C++, C, REXX comments (/* ... */) and blank lines are excluded from the comparison set. This option produces a listing with all comments removed and blanked.

Valid for LINE and WORD compare types and Search.

DPPSCMT Do not process Pascal-type comments. Comments of the type (* ... *) and blank lines are excluded from the comparison. DPPSCMT and DPPLCMT may be required for some Pascal compiler comments. This option produces a comparison listing with comments removed and part comments blanked.

Valid for LINE and WORD compare types and Search.

FMSTOP Immediately a difference is found between files, stops the compare with a return code of 1. This option provides a quicker way of telling if two files are different.

Valid for FILE compare type.

FMVLNS Flag moved lines. Identify inserted lines from the *new* file that match deleted lines from the *old* file. Inserted-moved lines are noted with "IM" and deleted-moved lines are noted with "DM" in the listing.

Valid for LINE compare type.

Notes:

1. Maximum length for lines is 256 characters.
2. Maximum length for a contiguous block of moved lines is 32K.

GWCBL Generates WORD/LINE comparison change bar listings. SuperC lists *new* file lines with change bars ("|") in column 1 for lines that differ between the *new* and *old* files. Deleted lines are indicated by flagging the lines following the deletion.

Valid for LINE and WORD compare types.

Notes:

1. LINE comparison and WORD comparison may give slightly different results due to their sensitivity to word and line boundaries. For further details, see "Reasons for Differing Comparison Results" on page 313.
2. GWCBL cannot be used with the process option Y2DTONLY.

IDPFX Identifier prefixed. File ID or member name is prefixed to the search string lines of the listing. See Figure 96 on page 289 for an example of a IDPFX listing.

Valid for Search.

LMCSFC Load module CSECT file compare list. Lists the name, number of bytes, and hash sum for each load module CSECT. Unchanged paired CSECTs are omitted when you specify the LOCS process option.

Notes:

1. LMCSFC is not supported for PDSE.
2. LMCSFC is supported on MVS only.

Valid for FILE compare type.

LMTO List group member totals. Lists the member summary totals and the overall summary totals for the entire file/group. See Figure 98 on page 290 for an example of an LMTO listing.

Valid for Search.

LNFMTO List "not found" member totals only. Lists the members that have no strings found for the entire file/group.

Valid for Search.

LOCS List only changed entries in summary. Normally, for *groups* of files/members being compared, SuperC lists all paired entries in the Member Summary Listing section of the listing file. Preceding the names of these pairs is a CHNG field to indicate whether the comparison found any differences or not. Figure 85 on page 281 shows a FILE comparison without LOCS. Figure 86 on page 282 shows a FILE comparison with LOCS.

When LOCS is specified, only those pairs which have changes are listed in the summary section.

Valid for group FILE, LINE, WORD, and BYTE compare types.

LONGLN Long lines. LONGLN causes SuperC to create a listing with 203 columns, reflecting up to 176 columns from the source files. This file may exceed the maximum number of columns handled by many printers.

Valid for LINE compare type and Search.

LPSF List previous-search-following lines. Lists the matched string line and up to 6 preceding and 6 following lines for context. The preceding and following count may be changed by using the LPSFV process statement. This allows a count range of 1 to 50 lines. A value of 0 is invalid, since this produces a normal search without any options.

Valid for Search.

LTO List totals only. List the overall summary totals for the entire file/member group. See Figure 100 on page 291 for an example of an LTO listing.

Valid for Search.

MIXED Mixed input. Indicates that the input text may be a mixture of both single-byte and double-byte (DBCS) text. Double-byte strings are recognized and handled differently than if MIXED were not specified. For instance, single byte characters are not valid within double-byte strings. Special terminal devices (for example, 5520) allow entry of DBCS characters.

Valid for LINE and WORD compare types.

NARROW Narrow side-by-side listing. Creates a 132/133 variable listing file with only 55 columns from each source file. Insertions and deletions are flagged and appear side-by-side in the listing output. Refer to Figure 82 on page 278 and Figure 83 on page 279 for examples of NARROW listings.

Valid for LINE comparison.

NOPRTCC No printer control columns. SuperC generates “normal” or NARROW listing files with record lengths of 133 columns, or WIDE or LONGLN listing with 203 columns. These listings contain printer control columns and page separators. NOPRTCC eliminates both the page separators and page header line. With NOPRTCC, “normal” and NARROW listings are 132 columns, and WIDE and LONGLN listings are 202. Section separators and title lines are still generated. This file may be preferred for on-line “browsing.”

Valid for FILE, LINE, WORD, and BYTE compare types and Search.

NOSEQ No Sequence numbers. Process fixed-length 80-byte record standard sequence number columns (73 to 80) as data. This option is extraneous for any record size other than 80.

Valid for LINE and WORD compare types and Search.

NOSUMS No Summary Section. Eliminates the group and final summary section from the output listing. This allows the user to generate a better “clean” copy for program inspection. Conversely, it eliminates the all-problem information in case of errors and option identification.

Valid for LINE, WORD, and BYTE compare types and Search.

REFMOVR Reformat override. Reformatted lines are not flagged in the output listing. They are, however, counted for the overall summary statistics and influence the return code since they are a special case of an insert/delete pair.

Valid for LINE compare type.

SDUPM Search duplicate members. Searches all members found in concatenated PDS data sets, even if more than one member is found to have the same name. Searches duplicate names even if the search is for a single member or if members are specified using the SELECT process statement.

Valid for Search.

Note: SDUPM is supported on MVS only.

SEQ Sequence numbers. Ignore fixed-length 80-byte record standard sequence number columns. Sequence numbers are assumed in columns 73 to 80 for such records. This option is invalid for any record size other than 80.

Valid for LINE and WORD compare types and Search.

SYSIN Provide alternate DD name for process statements. Syntax is `SYSIN(DDNAME)`. The default ddname is SYSIN. If this option is used, SuperC only accesses process statements via the supplied ddname. It does not attempt to access additional process statements via the SYSIN2 DD card.

Valid for FILE, LINE, WORD, and BYTE compare types and Search.

Note: SYSIN is supported on MVS only.

UPDCMS8 Update CMS 8 format. UPDCMS8 produces an update file that contains both control records and source lines from the *new* input file. UPDCMS8 requires that the *old* file has fixed-length 80-byte records with sequence numbers. The *new* file may have a variable or fixed length format with an LRECL ≤ 80.

SuperC may change the status of match lines to insert/delete pairs, enlarging the sequence number gaps of the *old* file. The update file (when properly named) can be used as input to CMS XEDIT. For information and an example of this update file, see “Update CMS Sequenced 8 File” on page 295.

Valid for LINE compare type.

UPDCNTL Update Control. Produces a control file which relates matches, insertions, deletions and reformattings using relative line numbers (for LINE compare type), relative word positions (for WORD compare type), or relative byte offsets (for BYTE compare type) within the *new* and *old* file. No source or data from either input file is included in the output file. “Do not” process options/statements are compatible selections for the LINE compare type. For information and an example of this update file, see “Update Control Files” on page 296.

Valid for LINE, WORD, and BYTE compare types.

UPDLDEL Update Long Control with all matches and delta changes. This reflects the comparison's matches, inserts, and deletes. You can edit this update file accepting, rejecting, or modifying the changes.

There are control records preceding each change and matching section. After the changes have been audited, optionally modified, and the control records removed, you should be able to reuse this control file as a composite new file.

Valid for LINE compare type.

UPDMVS8 Update MVS8 format. Produces a file that contains both control and *new* file source lines. Sequence numbers from columns 73 to 80 of the *new* file are used (when possible) as insert references, while deletes use sequence numbers from columns 73 to 80 of the *old* file. Both files must have fixed-length 80-byte records. The format of the generated data may be suitable as MVS IEBUPDTE input. For information and an example of this update file, see "Update MVS Sequenced 8 File" on page 300.

Valid for LINE compare type.

UPDPDEL Update prefixed delta lines. Produces a control data set containing header records and complete (up to 32K line length limit) delta lines from the input source files. Each output record is prefixed with identification and information. The update data set is a variable-length data set reflecting the input source files' characteristics.

Valid for LINE compare type.

UPDREV Update Revision. UPDREV produces a copy of the *new* file with SCRIPT/VS *.rc on/off* or IBM BookMaster *:rev/:erev* revision codes delimiting most script lines that contain changes.

You may wish to contrast the source lines delimited by the UPDREV option and a similar flagging of the lines with changes from the output listing file as produced by the GWCBL process.

Note: The revision character used is controlled by using the REVREF process statement. For details, see "Revision Code Reference" on page 255.

A REVREF process statement (for example, `REVREF REFID=ABC` or `REVREF RCVAL=1`) defines the revision level (SCRIPT/VS tags) or reference ID (IBM BookMaster tags). Alternatively, SCRIPT/VS .rc delimiters may be controlled by the first record in the *new* file. (For example, `.rc 2 |` as the first record causes level 2 to be used).

Note: IBM BookMaster requires the REFID value to be defined with a *:revision* tag and "RUN=YES" attribute to have the change character inserted in the processed document.

For information and an example of this update file, see "Revision File" on page 293.

Valid for LINE and WORD compare types.

UPDREV2 Update Revision (2). UPDREV2 is identical to UPDREV with the exception that data between the following IBM BookMaster tags are not deleted in the update file:

```
:cgraphic.
:ecgraphic.

:fig.
:efig.

:lblbox.
:elblbox.

:nt.
:ent.
```

```
:screen.
:escreen.

:table.
:etable.

:xmp.
:exmp.
```

Valid for LINE and WORD compare types.

UPDSEQ0 Update Sequence 0 (zero). UPDSEQ0 produces a control file that relates insertions and deletions to the relative line numbers of the *old* file. Both control records and *new* file source lines are included in the output file. This option is similar to UPDCMS8 except that it uses relative line numbers (starting with zero) instead of the sequence numbers from columns 73 to 80. The control field after a "$" designates the number of new source lines that follow in the update file.

Both fixed and variable record length lines are allowed. Fixed-length records shorter than 80 bytes are padded with spaces to 80. Insertion lines are full fixed or variable length copies of the new input data set lines. For information and an example of this update file, see "Update Sequenced 0 File" on page 301.

Valid for LINE compare type.

UPDSUMO Update Summary only. UPDSUMO produces an update file of 4 lines (new file ID, old file ID, totals header, single summary line). The summary line is tagged with a "T" in column 1 and the summary statistics are located at fixed offsets in the output line. The file has a record length of 132. For information and an example of this update file, see "Update Summary Only Files" on page 302.

Valid for LINE, WORD, and BYTE compare types.

WIDE Wide side-by-side listing. Creates a 202/203 variable-length listing file with 80 columns from each source file. Inserts and deletes are flagged and appear side-by-side in the listing output. For an example of a WIDE side-by-side listing, see Figure 84 on page 280.

Valid for LINE compare type.

XREF Cross reference strings. Creates a cross reference listing by search string. Can be used with IDPFX, LMTO LNFMTO, and LTO. Not implemented for LPSF.

The XREF option can be useful when more than one search string (or search condition) is specified. The XREF listing is implemented using a multiple pass operation for listing the "lines found" for each individual string. Be aware that XREF adds some additional processing overhead to the normal search process. For an example of a search XREF listing, see Figure 92 on page 286.

Valid for Search.

XWDCMP Extended WORD comparison. The word delimiter set is extended to include non-alphanumeric characters (including spaces). For example, “ABCD(EFGH) JKL” is 2 words using normal WORD compare type, but 5 (3 words and 2 pseudo-words) with the XWDCMP process option.

Valid for WORD compare type.

Y2DTONLY Compare Dates Only. Indicates that the comparison process is to be performed only on the dates defined by the Date Definition process statements. That is, all data in the input files is ignored in the comparison process apart from that defined by NY2C, NY2Z, NY2D, NY2P, OY2C, OY2Z, OY2D, and OY2P process statements. For further details on these process statements, see “Date Definitions” on page 263.

Notes:

1. Y2DTONLY causes a “record-for-record” comparison to be performed between the two input files, whereby dates are simply checked for being equal or unequal. (The “high/low” comparison logic that SuperC normally uses is not applied in the case of Y2DTONLY and, as such, the relative *values* of the dates have no bearing on the result of the comparison.)
2. Y2DTONLY is not supported for the process option GWCBL (change bar listing).

Valid for LINE compare type.

Process Statements

You can use process statements to tailor your comparison or search according to your requirements. Process statements provide a powerful and flexible way of ensuring that only relevant data is compared (or searched) and that meaningful results are produced.

Broadly speaking, the two major functions that process statements perform are:

- To *select* the data that is to be compared (or searched) and,
- To handle various *date formats*.

All process statements require a keyword followed by one or more operands. They are supplied to SuperC in the Process Statements File.

Table 26 on page 237 lists each of the process statement keywords and shows for which compare type each keyword can be used. The table also shows whether the keyword is valid for the SuperC Search.

Note: The sequence in which each of the process statements is listed (in Table 26 on page 237 and the pages following) is primarily alphabetic according to the process statement keyword.

However, in the interest of keeping associated “pairs” and “sets” of process statements together, the prefixes “N” and “O” (indicating the process statement applies to the new or old file respectively) have been ignored when sequencing the process statements alphabetically.

Similarly, the three process statements NEWDD, OLDDD, and UPDDD have been kept together and sequenced according to the “DD” portion of the keyword.

Table 26 (Page 1 of 2). Summary of Process Statements

Process Statement		Valid For				
		Compare Type				Search
Keyword	**Description**	**FILE**	**LINE**	**WORD**	**BYTE**	
NCHGT	Change text: new or search file		√	√		√
OCHGT	Change text: old file		√	√		
CHNGV	Change listing value		√	√	√	
CMPBOFS	Compare byte offsets				√	
CMPCOLM	Compare (search) columns: new, old, search files		√	√		√
CMPCOLMN	Compare columns: new file		√	√		
CMPCOLMO	Compare columns: old file		√	√		
CMPLINE	Compare lines		√	√		√
CMPSECT[1]	Compare sections		√	√		
COLHEAD[2]	Define column headings		√			
NEWDD[3]	VSE DLBL/TLBL Definition: new file, or MVS alternate DDNAME	√	√	√	√	√
OLDDD[3]	VSE DLBL/TLBL Definition: old file, or MVS alternate DDNAME	√	√	√	√	
UPDDD[3]	VSE DLBL/TLBL Definition: update file, or MVS alternate DDNAME	√	√	√	√	
DPLINE	Do not process lines (containing a string)		√	√		√
DPLINEC	Do not process lines continuation		√	√		√
NEXCLUDE[4]	Exclude data: new file	√	√			
OEXCLUDE[4]	Exclude data: old file	√	√			
NFOCUS[4]	Focus on data: new file	√	√			
OFOCUS[4]	Focus on data: old file	√	√			
LNCT	Line count	√	√	√	√	√
LPSFV	List previous-search-following value					√
LSTCOLM	List columns		√			√
REVREF	Revision code reference		√	√		
SELECT	Select PDS members (MVS)	√	√	√	√	√
SELECT	Select members/files (CMS)	√	√	√	√	√
SELECT	Select members (VSE)	√	√	√	√	√
SELECTF[5]	Select files from a list	√	√	√	√	√
SLIST	Statements listing option	√	√	√	√	√
SRCHFOR	Search for a string					√
SRCHFORC	Search for a string continuation					√
NTITLE	Alternative listing title: new file	√	√	√	√	√
OTITLE	Alternative listing title: old file	√	√	√	√	
NY2AGE	Aging option: new file		√			
OY2AGE	Aging option: old file		√			
NY2C	Date definition: new file, character format		√			
NY2Z	Date definition: new file, zoned decimal format		√			

Table 26 (Page 2 of 2). Summary of Process Statements

Process Statement		Valid For				
		Compare Type				Search
Keyword	**Description**	**FILE**	**LINE**	**WORD**	**BYTE**	
NY2D	Date definition: new file, unsigned packed decimal format		√			
NY2P	Date definition: new file, packed decimal format		√			
OY2C	Date definition: old file, character format		√			
OY2Z	Date definition: old file, zoned decimal format		√			
OY2D	Date definition: old file, unsigned packed decimal format		√			
OY2P	Date definition: old file, packed decimal format		√			
WORKSIZE	Maximum number of units for comparison		√	√	√	
Y2PAST	Global date option		√			
*	Process Statement comment to be printed	√	√	√	√	√
.*	Process Statement comment not to be printed	√	√	√	√	√

Note:

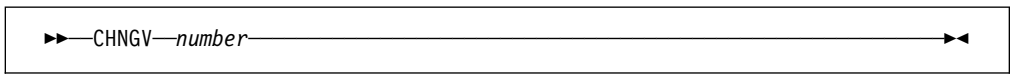
1. Not supported on CMS.
2. Valid only for listing types DELTA and LONG.
3. Supported only on VSE.
4. FILE compare type is valid only with ROWS option of NEXCLUDE, OEXCLUDE, NFOCUS and OFOCUS.
5. Supported only on CMS.

The following sections describe each process statement in detail.

Change Listing Value

The CHGNV process statement specifies the number of match lines listed before and after a line with a change: insert, delete, or reformat.

Compare Types: LINE, WORD, and BYTE



number A decimal number between 1 and 1000.

Example	Description
`CHNGV 3`	Lists up to 3 lines before and after change.

Change Text

There are two Change Text process statements:

NCHGT Change new (or search) input text string

OCHGT Change old input text string

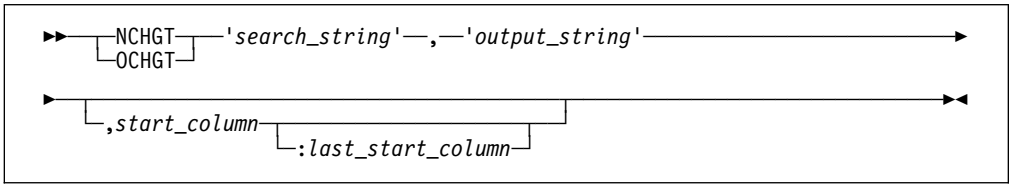
These process statements change the input source image before performing the comparison.

The relative input file (“new” or “old”) is scanned for text that matches a *search_string*. If matching text is found, it is replaced by a corresponding

output_string before the comparison process is performed. Question marks (“?”) may be used as “wildcard” characters in the *search_string* or *output_string*.

The *search_string* and *output_string* need not be the same length. The *output_string* may even be a null string.

Compare Types: LINE, WORD, and Search
OCHGT cannot be used for Search.



search_string A character or hexadecimal string to be replaced in the input file. For imbedded single quotes, use two consecutive single quotes ('').

output_string The replacement string to be used in the comparison. For imbedded single quotes, use two consecutive single quotes ('').

start_column The column in or after which the *search_string* must start. Must be greater than zero.

last_start_column The last column in which the *search_string* may start. Must be separated from the *start_column* by a colon, and must be equal to or greater than the *start_column* value. If not supplied, is the equivalent of setting the value to *start_column*. To search from the *start_column* to the end of a variable length row, set the ***last_start_column*** to a value larger than the length of the longest row.

Example	Description
`NCHGT 'ABCD','XXXX'`	Changes all strings “ABCD” in the new file to “XXXX” before performing the comparison.
`OCHGT 'ABCD','XXXX',1:50`	Changes all strings “ABCD” in the old file, that start within columns 1 to 50, to “XXXX” before performing the comparison.
`OCHGT 'ABCD','',1:50`	Changes all strings “ABCD” in the old file, that start within columns 1 to 50, to a null string before performing the comparison. (In the comparison process, this effectively ignores any “ABCD” strings found in those positions in the old file.)
`NCHGT 'ABCD','AB'`	Changes all strings “ABCD” in the new file to “AB” before performing the comparison.
`NCHGT X'7B01',':1',6`	Changes all hexadecimal strings `X'7B01'` in the new file, that start in column 6, to the character string “:1” before performing the comparison.
`NCHGT 'PREF???','NPREF'`	Changes all 7-character strings with the prefix “PREF” in the new file, to the 5-character string “NPREF” before performing the comparison.
`NCHGT 'PREF???','NPREF??'`	Changes the first 5 characters of all 7-character strings with the prefix “PREF” in the new file, to “NPREF” before performing the comparison.

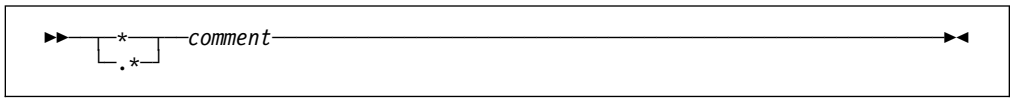
Comment Lines

There are two tags that if found at the start of a line turn it into a comment line:

`*` An asterisk as the first character on a process statement line begins a printable comment line.

`.*` A period-asterisk as the first two characters on a process statement line begins a comment that is not printed in the SuperC listing.

Compare Types: FILE, LINE, WORD, BYTE, and Search



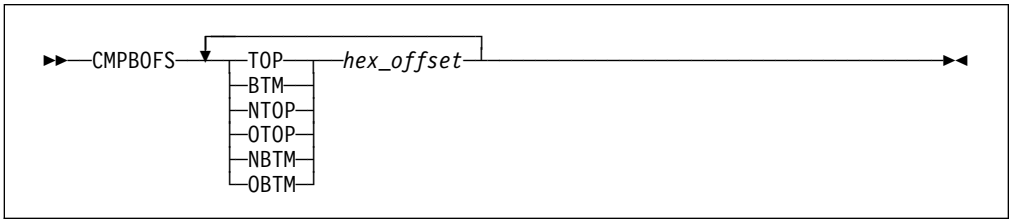
***** Must be in column 1.
.* Must be in columns 1 and 2.

Example	Description
`*`	This comment prints in the SuperC listing.
`.*`	This comment does not print in the SuperC listing.

Compare Byte Offsets

The CMPBOFS process statement compares a file between byte limits. The start and stop reference values must be hex values. The statement may be specified on one complete line or may have separate CMPBOFS statements for each of the six keyword operands: TOP, BTM, NTOP, NBTM, OTOP, and OBTM.

Compare Type: BYTE



keyword The keyword may be one of the following:

TOP Top. Defines the first byte offset position in the *new* and *old* byte compare file. Means both NTOP and OTOP. The lowest byte position is at offset zero.

NTOP New Top. Defines the first byte offset position in the *new* file for the byte compare.

OTOP Old Top. Defines the first byte offset position in the *old* file for the byte compare.

BTM Bottom. Defines the last byte position in the *new* and *old* byte compare file. Means both NBTM and OBTM.

NBTM New Bottom. Defines the ending point in the *new* file for the compare.

OBTM Old Bottom. Defines the ending point in the *old* file for the compare.

| ***hex_offset*** A hexadecimal value. Do not put in quotes, or 'bracket' it within "X'...'."

Example	**Description**
`CMPBOFS NTOP 1000 OTOP 5E00`	Compare the new file from hex offset `X'1000'` (to the end of file) with the old file from hex offset `X'5E00'` (to the end of file).
`CMPBOFS NTOP 1000` `CMPBOFS OTOP 5E00`	These two separate process statements have the same effect as the "combined" statement above.

Compare (Search) Columns

There are three Compare Columns process options:

CMPCOLM Applies to both the new and old files, or search file

CMPCOLMN Applies to the new file

CMPCOLMO Applies to the old file

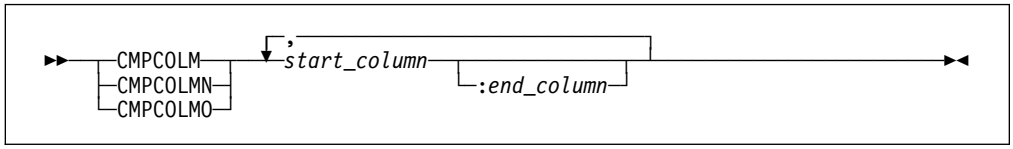
These options compare (or search) the data between column limits of the input files (or search file). Up to 15 compare ranges or individual columns are allowed and

may be entered on additional CMPCOLM, CMPCOLMN, or CMPCOLMO statements. All specified ranges of columns must be in ascending order.

Compare Types: LINE and WORD
CMPCOLM is also valid for Search.

Notes:

1. Some process options (SEQ, NOSEQ, and COBOL) also specify columns. The CMPCOLM, CMPCOLMN, CMPCOLMO process statements override all of these process options.
2. CMPCOLM, CMPCOLMN, CMPCOLMO cannot be used for WORD compare type or Search if the input contains a *mixture* of DBCS and non-DBCS data.



start_column The starting column number to be compared or searched.
end_column The ending column number of the range of columns to be compared or searched. (Must be separated from the *start_column* by a colon.)

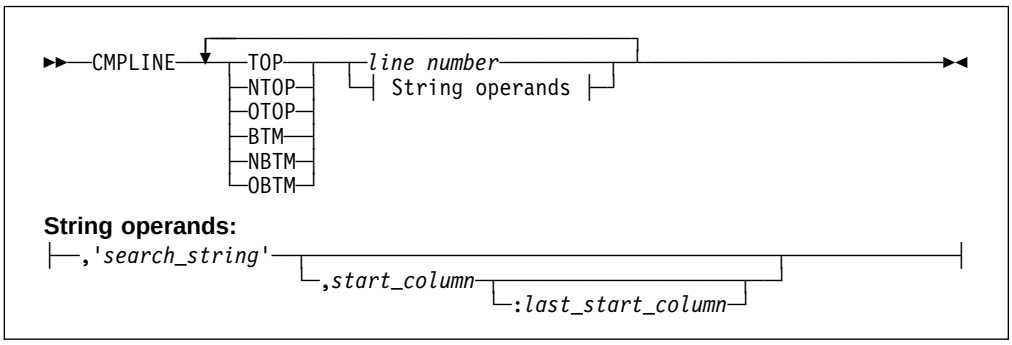
Example	Description
`CMPCOLM 25:75`	Compare columns 25 through 75 in both files (or search columns 25 through 75 in the search file).
`CMPCOLM 30:60,75`	Compare columns 30 through 60 and column 75 in both files (or search columns 30 through 60 and column 75 in the search file).
`CMPCOLMN 48:54`	Compare columns 48 through 54 in the new file.
`CMPCOLMO 87`	Compare column 87 in the old file.
`CMPCOLMN 17:22` `CMPCOLMO 15:20`	Compare columns 17 through 22 in the new file with columns 15 through 20 in the old file.

Compare Lines

The CMPLINE process statement compares two files (or search) between line limits. The statement may be specified on one complete line or may have separate CMPLINE statements for each of the six keyword operands: TOP, BTM, NTOP, NBTM, OTOP, and OBTM. The reference values may be line numbers or data strings.

Compare Types: LINE, WORD, and Search

Note: Keyword operands OTOP and OBTM are invalid for Search.



keyword The keyword may be one of the following:

TOP Top. Defines the beginning line in the *new* (or search) file and *old* compare file. Means both NTOP and OTOP.

NTOP New Top. Defines the beginning line in the *new* (or search) file.

OTOP Old Top. Defines the beginning line in the *old* file.

BTM Bottom. Defines the ending line in the *new* (or search) file and *old* compare file. Means both NBTM and OBTM.

NBTM New Bottom. Defines the ending line in the *new* (or search) file.

OBTM Old Bottom. Defines the ending line in the *old* compare file.

line number The relative number of the record in the file.

search_string A character or hexadecimal string enclosed within single quotes. For imbedded single quotes, use two consecutive quotes ('').

start_column The column in or after which the *search_string* must start.

last_start_column The last column in which the *search_string* may start. Must be separated from the *start_column* by a colon.

Example	Description
`CMPLINE TOP 55 BTM 99`	Compare from line 55 to line 99 in both files.
`CMPLINE NTOP 55 NBTM 99`	Compare from line 55 to line 99 in the new file.
`CMPLINE NTOP 'ABCD',5:66`	Compare from where “ABCD” starts within columns 5 to 66 in new file (that is, is found within columns 5 to 69).
`CMPLINE OTOP 'ABCD'`	Compare from where “ABCD” first found in old file.
`CMPLINE TOP X'40E2',1:1`	Compare from where “ S” is found for both files.

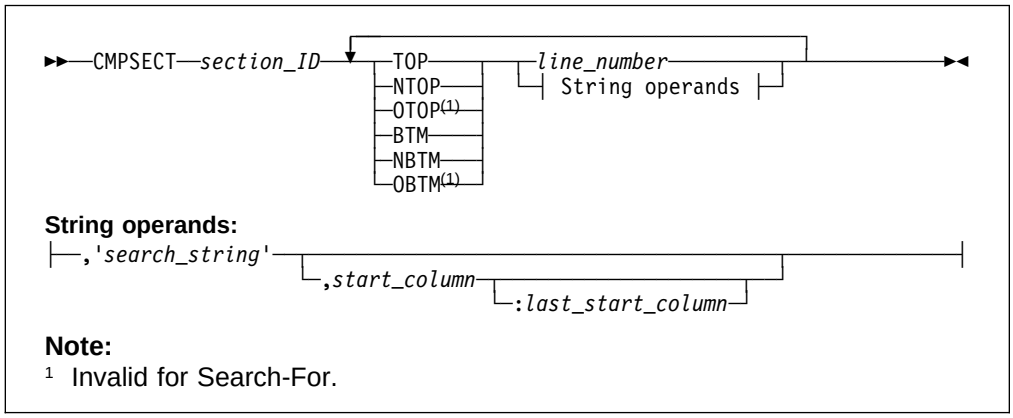
Compare Sections

The CMPSECT process statement compares multiple sections from one sequential data set or PDS member to another sequential data set or PDS member. It is not valid for a PDS group comparison of more than one member. It is functionally similar to CMPLINE but allows you to divide the input into one or more *sections* for subsequent comparison or searching. A section ID name is needed to associate all keyword operands to a particular section. Thus, multiple sections of the input can be compared (or searched) in a single execution of SuperC.

Compare Types: LINE, WORD, and Search

Notes:

1. CMPSECT is not supported for CMS.
2. Keywords OTOP and OBTM are invalid for Search.



section_ID A character string identifier (up to 8 alphanumeric characters, no imbedded spaces, can start with a numeric) relating to a section (group of lines). It allows multiple keywords to be associated with the same section.

keyword The keyword may be one of the following:

- **TOP** Top. Defines the beginning line in the *new* (or search) file and *old* compare section. Means the same as NTOP and OTOP.
- **NTOP** New Top. Defines the beginning line in the *new* (or search) section.
- **OTOP** Old Top. Defines the beginning line in the *old* section.
- **BTM** Bottom. Defines the ending line in the *new* (or search) file and *old* compare section. Means both NBTM and OBTM.
- **NBTM** New Bottom. Defines the ending line in the *new* (or search) section.
- **OBTM** Old Bottom. Defines the ending line in the *old* compare section.

line_number The line number associated with the **keyword**.

string A character or hexadecimal string enclosed within single quotes. For imbedded single quotes, use two consecutive quotes (' ').

start_column The column in or after which the *search_string* must start.

last_start_column The last column in which the *search_string* may start. Must be separated from the *start_column* by a colon.

Note: If a "top" condition is not found (for example, a pattern is incorrect), the compare continues but usually reports zero lines processed for this data set.

Example	Description
`CMPSECT SECT01 TOP 25 BTM 50`	Compares lines 25 through 50 in both data sets or members.
`CMPSECT SECT02 NTOP 60 NBTM 70` `CMPSECT SECT02 OTOP 65 OBTM 75`	Compares lines 60 through 70 in the new data set to lines 65 through 75 in the old data set.
`CMPSECT SECTX TOP 'PART1:',2:10` `CMPSECT SECTX BTM 'END PART1:',2:10`	Starts the comparison of both data sets when SuperC detects the string "PART1:" starting in columns 2 through 10 and ends the comparison when SuperC detects the string "END PART1:" starting in columns 2 through 10.

`CMPSECT SECTY NTOP 'PART2:',2:10`
`CMPSECT SECTY OTOP 'PART2:',6:20`
`CMPSECT SECTY BTM 'END PART2:',2:10`
	Compares a section in the new data set to a section in the old data set. The section in the new data set begins with the string "PART2:" in columns 2 through 10 and ends with the string "END PART2:" in columns 2 through 10. The section in the old data set begins with the string "PART2:" in columns 6 through 20 and ends with the string "END PART2:" in columns 2 through 10.

Note: All of the previous statements could be combined to compare multiple sections of the new and old data sets.

DD-MVS Alternate DD Names

There are three DD-MVS Alternate DD Names process statements:

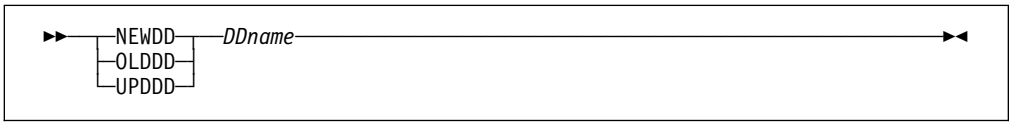
NEWDD Name applies to the new (or search) file

OLDDD Name applies to the old file

UPDDD Name applies to the update file

These process statements allow you to specify alternative names for the new and old input files, and for the output update file. (The default names are NEWDD, OLDDD, and UPDDD respectively.)

Compare Types: FILE, LINE, WORD, and BYTE.
NEWDD is also valid for Search.



DDname The name of the DD card to be processed.

Example	Description
`UPDDD FILE3`	Update file is refererenced via DD card with the name FILE3.
`NEWDD FILE4`	Name of new file DD is FILE4.

DD-VSE DLBL/TLBL Definitions

There are three DD-VSE DLBL/TLBL Definitions process statements:

NEWDD Name applies to the new (or search) file

OLDDD Name applies to the old file

UPDDD Name applies to the update file

For VSE sequential files, these process statements allow you to specify:

- Alternative names for the *new* and *old* input files, and for the output update file. (The default names are NEWDD, OLDDD, and UPDDD respectively.)
- The file attributes for the *new* and *old* input files. (The default attributes are: non-VSAM, fixed, unblocked, record size 80).

For VSE Librarian members, these process statements allow you to select members from the *new* and *old* sublibraries.

Notes:

1. NEWDD, OLDDD, and UPDDD process statements as discussed in this section apply to VSE only. See also “DD-MVS Alternate DD Names” on page 245.
2. When NEWDD is used in the SuperC Search, references to the “new” file in the following pages indicate the *search* file.
3. For more information on the Job Control Language (JCL) required for the new (or search), old, and update files, see “Invoking the Comparison on VSE” on page 199 and “Invoking the Search on VSE” on page 221.

VSE (Disk) Files: If the input is a SAM file not managed by VSAM, the BLKSIZE, RECSIZE, and RECFORM values are required (otherwise the default file attributes apply).

In the case of a VSAM-managed SAM file, the file attributes are normally checked for in the VSAM catalog. However, if the attributes are supplied via this statement, the VSAM catalog definitions are ignored.

For native VSAM files (KSDS, ESDS, RRDS, VRDS), the catalog attributes are always used.

For fixed files, the block size, record size, and record format are used for deblocking and memory allocation.

For variable files, the blocksize allocates enough memory to hold a full block, and the record size is not required.

The file attributes for the output update file are determined by the type of *update* process option that is used (see “Process Options” on page 226).

VSE (Tape) Files: If the input is a tape file, it can have standard labels or it can be unlabeled. If the file attributes are not supplied via the NEWDD or OLDDD statement, they are assumed to be: fixed length, unblocked, record size 80.

VSE Librarian Members: When using the NEWDD or OLDDD process statement for VSE libraries, the library and sublibrary must be defined.

The NEWDD and OLDDD process statements can be used to:

- Specify *just* the Librarian library and sublibrary, then:
  - Select individual members by using SELECT process statements (for further details, see "Select Members (VSE)" on page 259).
  - Select the whole sublibrary (by *not* using SELECT process statements).
- Select an individual member by specifying the Librarian library, sublibrary, member name, and member type.
- Select a *group* of members by using the wildcard character "*" (asterisk) in either the member name or the member type (or both). See the following for a more detailed description of the way in which *groups* of members can be selected.

Using the Wildcard Character to Select Groups of Members: You can use a "*" as a generic indicator as part of the member name or member type (or both) to select a *group* of members for subsequent input to either the SuperC Comparison or the SuperC Search. The "*" may only be used at the beginning (prefix) or the end (suffix) of the member name or member type.

Examples:

Member Name	Member Type	Members Selected (Member Name and Type shown only)
MEM1	TYPA	MEM1.TYPA (Selection of a single member)
MEM*	TYPA	All members with a name starting with "MEM" and a type of "TYPA"
MEM*	*	All members with a name starting with "MEM" regardless of type
*MEM	TYP*	All members with a name ending with "MEM" and a type starting with "TYP"
MEM	*TYP*	Invalid use of wildcard character; can only be used as a prefix *or* a suffix to the same item
*	*	All members (in the sublibrary of the Librarian library)

Note: When using the NEWDD and OLDDD process statements with wildcard ("*") characters to select a *new group* of members for comparison with an *old group* of members, be aware of the way in which individual members (within each group) are "paired" by SuperC for subsequent comparison.

SuperC "pairs" members from each (sorted) group:

- According to the portion of the member name or member type which was represented by the "*" wildcard character (when SuperC initially selected the members for inclusion in the group), and
- Ignoring the remainder of the member name or member type

For example, if you specified a member name of `ABC1*` in the NEWDD process statement (to select all *new* members with names starting with "ABC1"), and a member name of `XYZ*` in the OLDDD process statement (to select all *old* members with names starting with "XYZ"), it *could* result in:

library.sublibrary.ABC11.*type*

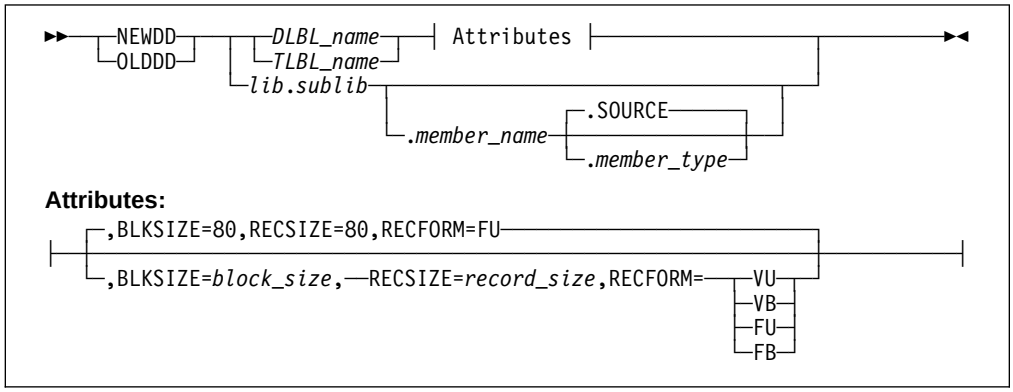
being compared with

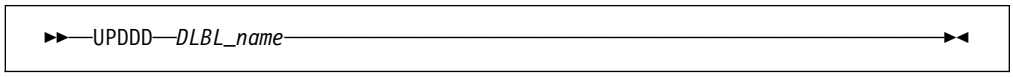
library.sublibrary.XYZ1.*type*

which may not be what you wanted.

If you find that members are not being "paired" as you want, use SELECT process statements to specify each individual pair of members that you want compared.

Compare Types: FILE, LINE, WORD, and BYTE.
NEWDD is also valid for Search.





DLBL_name Your own choice of DLBL name for the new, old, or update file.

TLBL_name Your own choice of TLBL name for the new or old file.

lib.sublib Library and sublibrary names. (Librarian members only.)

member_name Name of the member in the sublibrary.

member_type Member type of the member in the sublibrary.

block_size The block size of the *new* or *old* file.

record_size The record size of the *new* or *old* file.

Note: For variable-length records, this must be the *maximum* record length.

RECFORM= The record format of the *new* or *old* file:

VU Variable, unblocked
VB Variable, blocked
FU Fixed, unblocked
FB Fixed, blocked

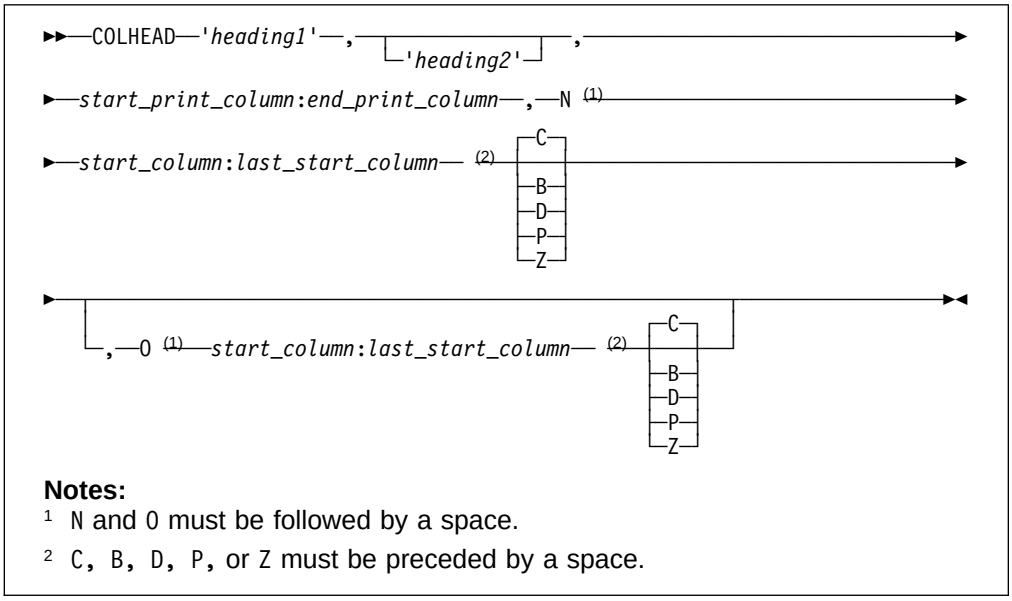
Example	Description
`NEWDD FILE1,BLKSIZE=160,RECSIZE=80,RECFORM=FB`	Name of *new* file is FILE1 with a block size of 160, and fixed blocked records of length 80.
`OLDDD FILE2,BLKSIZE=120,RECSIZE=120,RECFORM=FU`	Name of *old* file is FILE2 with fixed unblocked records of length 120.
`UPDDD FILE3`	Name of update file is FILE3.
`NEWDD FILE4`	Name of *new* file is FILE4 with (default file attributes) fixed unblocked records of length 80.
`NEWDD MAINLIB.LIBA.MEMB1.C`	Selects *new* member MEMB1 (with a member type of C) in sublibrary LIBA, in library MAINLIB.
`OLDDD MAINLIB.LIBA.MEMB*.C`	Selects all *old* members with a member name starting with “MEMB” (and with a member type of C) in sublibrary LIBA, in library MAINLIB.
`NEWDD MAINLIB.LIBA.*.C`	Selects all *new* members with a member type of C in sublibrary LIBA, in library MAINLIB.

Define Column Headings

The COLHEAD process statement defines column headings and specifies the location and format of the corresponding data to be displayed. For an example of a listing with column headings, see Figure 81 on page 277.

Note: COLHEAD is not available for side-by-side listings. (See “NARROW” process option on page 232).

Compare Type: LINE



Notes:

1 `N` and `O` must be followed by a space.

2 `C`, `B`, `D`, `P`, or `Z` must be preceded by a space.

heading1 The heading to appear on the first line for the print column.

heading2 The heading to appear on the second line for the print column.

start_print_column The starting print column for the heading specified.

end_print_column The ending print column for the heading specified. (Must be separated from the *start_print_column* by a colon.)

Note: If the print-column range is shorter than the heading specified, the heading is truncated.

N Indicates the operands following relate to the *new* file.

start_column The starting position in the *new* file of the data to be displayed.

last_start_column The ending position in the *new* file of the data to be displayed. (Must be separated from the *start_column* by a colon.)

Data Format Indicator The format of the data in the *new* file to be displayed:

C	Character
B	Binary
D	Unsigned packed decimal
P	Packed decimal
Z	Zoned decimal

O Indicates the operands following relate to the *old* file.

start_column The starting position in the *old* file of the data to be displayed.

last_start_column The ending position in the *old* file of the data to be displayed. (Must be separated from the *start_column* by a colon.)

Data Format Indicator The format of the data in the *old* file to be displayed (as for the *new* file).

Example

```
COLHEAD 'START','DATE',1:7,N 1:6 P,O 11:16
```

Description

Defines a print column with a heading of “START” in the first line and “DATE” in the second heading line, headings to start in print column 1. The data to be displayed from the new file is in positions 1 through 6 and is in packed format. The data to be displayed from the old file is in positions 11 through 16 and is in (the default) character format.

Do Not Process Lines

There are two Do Not Process Lines process statements:

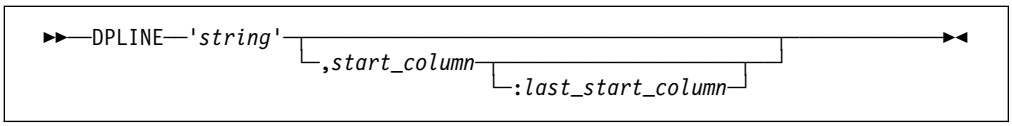
DPLINE Do not Process Lines

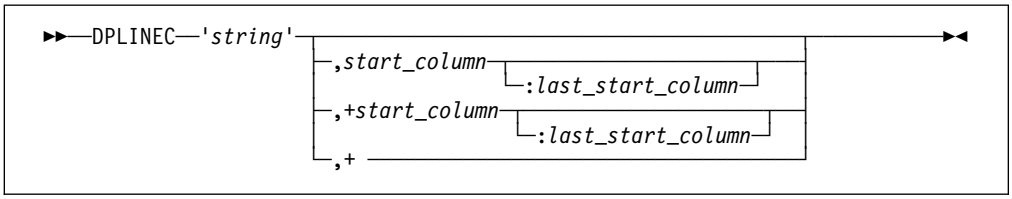
DPLINEC Do not Process Lines Continuation

These options remove from the compare (or search) set all lines that can be recognized by either a unique character string or combination of related strings all appearing on the same input line. DPLINEC is the continuation of the immediately preceding DPLINE or DPLINEC process statement. All the strings in a DPLINE/DPLINEC group must be found on the same input line.

A start_column or start-range can also be used to restrict the columns. Relative start_columns and start-ranges are valid only on DPLINEC statements.

Compare Types: LINE, WORD, and Search





string A character or hexadecimal string enclosed within single quotes. For imbedded single quotes, use two consecutive quotes ('').

start_column The column in, or after which, the string must start.

last_start_column The last column in which the string may start. (Must be separated from the *start_column* by a colon.)

+start_column The relative column, following the location of the previous string (as specified in the previous DPLINE or DPLINEC statement), in, or after which, this string must start.

last_start_column The relative last column, following the location of the previous string (as specified in the previous DPLINE or DPLINEC statement), in which this string may start.

+ The specified string may appear anywhere following the location of the previous string (as specified in the previous DPLINE or DPLINEC statement).

Example	Description
`DPLINE 'ABCDE'`	Scans all columns for string “ABCDE”
`DPLINE 'AbCde',2`	Scans only column 2 for start of string “AbCde”
`DPLINE 'AbCde',2:2` `DPLINEC 'BDEF'`	Same as above example. String “BDEF” must be on the same line as the string “AbCde”
`DPLINE 'ABCDE',2:50`	Scans only columns 2 through 50 for start of string “ABCDE”
`DPLINE 'AB''CD',2:50`	Scans only columns 2 to 50 for start of string “AB'CD”
`DPLINE X'C1C27BF1',2:50`	Scans only columns 2 to 50 for start of hexadecimal string X'C1C27BF1'
`DPLINE 'ABC'` `DPLINEC 'BDEF',+`	Scans for string “ABC”; if found, then scans for string “BDEF” in the same line (following “ABC”)
`DPLINE 'ABC'` `DPLINEC 'BDEF',+5`	Scans for string “ABC”; if found, then scans for string “BDEF” starting in the 5th column after the starting column of “ABC”
`DPLINE 'ABC'` `DPLINEC 'BDEF',+5:12`	Scans for string “ABC”; if found, then scans for string “BDEF” starting anywhere in the 5th to 12th columns after the starting column of “ABC”

Exclude Data

There are two Exclude Data process statements:

NEXCLUDE Exclude applies to the new file

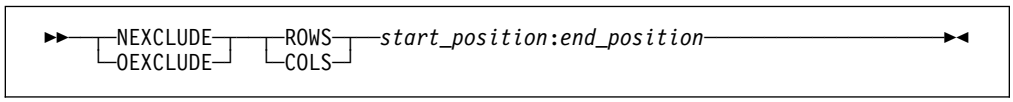
OEXCLUDE Exclude applies to the old file

These statements exclude rows or columns of data from the comparison. Up to 254 “exclude” statements can be entered for each file.

Notes:

1. NEXCLUDE and OEXCLUDE statements are mutually exclusive to NFOCUS and OFOCUS statements respectively if using the same operand keyword (ROWS or COLS).
2. Do not use the NEXCLUDE or OEXCLUDE process statement if the Y2DTONLY process statement has been specified.

Compare Types: FILE (ROWS option only) and LINE



start_position If ROWS operand used, the first row (record) to be excluded from the comparison.
If COLS operand used, the first column to be excluded from the comparison.

end_position If ROWS operand used, the last row (record) to be excluded from the comparison.
If COLS operand used, the last column to be excluded from the comparison.
(Must be separated from the *start_position* by a colon.)

Example	Description
`NEXCLUDE ROWS 5:900`	Excludes rows (records) 5 through 900 on the new file.
`OEXCLUDE ROWS 1:900`	Excludes rows (records) 1 through 900 on the old file.
`OEXCLUDE COLS 100:199`	Excludes columns 100 through 199 on the old file.

Focus on Data

There are two Focus on Data process statements:

NFOCUS Focus applies to the new file

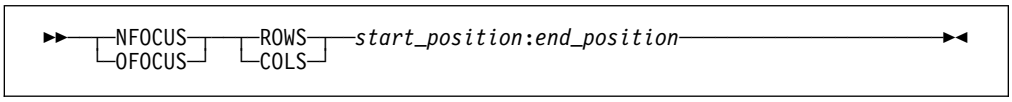
OFOCUS Focus applies to the old file

These two statements select (or “focus on”) rows or columns of data to be compared. In other words, only these rows or columns are considered when performing the comparison (or search) process and all other rows or columns are ignored. Up to 254 “focus” statements can be entered for each file.

Notes:

1. NFOCUS and OFOCUS statements are mutually exclusive to NEXCLUDE and OEXCLUDE statements respectively if using the same operand keyword (ROWS or COLS).
2. Do not use the NFOCUS or OFOCUS process statement if the Y2DTONLY process statement has been specified.

Compare Types: FILE (ROWS option only) and LINE



start_position If ROWS operand used, the first row (record) to be selected for the comparison.
If COLS operand used, the first column to be selected for the comparison.

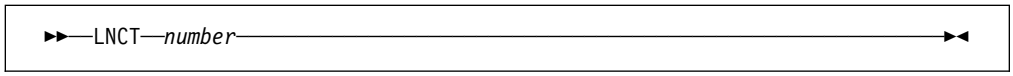
end_position If ROWS operand used, the last row (record) to be selected for the comparison.
If COLS operand used, the last column to be selected for the comparison.
(Must be separated from the *start_position* by a colon.)

Example	Description
`NFOCUS ROWS 28:90`	Selects rows (records) 28 through 90 on the new file.
`OFOCUS ROWS 150:165`	Selects rows (records) 150 through 165 on the old file.
`OFOCUS COLS 10:18`	Selects columns 10 through 18 on the old file.

Line Count

The LNCT process statement specifies the number of lines per page in the listing file.

Compare Types: FILE, LINE, WORD, BYTE, and Search



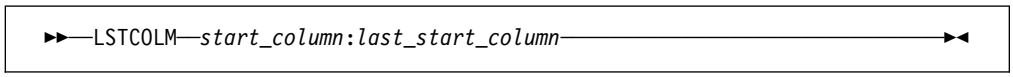
number A decimal number between 15 and 999999.

Example	Description
`LNCT 55`	Lists up to 55 lines per page.

List Columns

The LSTCOLM process statement selects a range of columns to be listed in the output. This statement overrides the defaults that SuperC generates. Column selections must be contiguous and can be no wider than the output listing line allocated (55/80/106/176).

Compare Types: LINE and Search



start_column The starting column to be listed.

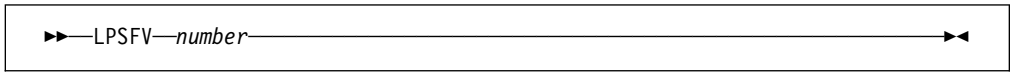
last_start_column The ending column to be listed. (Must be separated from the *start_column* by a colon.)

Example	Description
`LSTCOLM 275:355`	Lists columns 275 through 355 in the output.

List Previous-Search-Following Value

The LPSFV process statement specifies the number of lines preceding and following the search line found to be listed. The default value is 6.

Compare Type: Search



number A decimal number between 1 and 50.

Example	Description
`LPSFV 2`	Lists up to 2 lines before and after the line found.

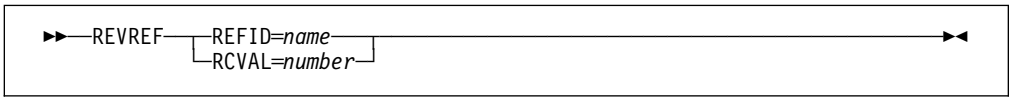
Revision Code Reference

The REVREF process statement identifies the revision type (IBM BookMaster or SCRIPT/VS) and level-ID for delimiting UPDREV and UPDREV2 output changes. The revision delimiter may, alternatively, be specified or indicated by using a SCRIPT/VS `.rc` definition statement as the first line of the *new* input file.

If either the UPDREV or UPDREV2 process option is specified and no REVREF process statement is in the statements file, or the first *new* file source line is not a `.rc` script definition statement, SuperC defaults the revision definition to a SCRIPT/VS specification of `.rc 1 |`.

Note: IBM BookMaster requires the REFID value to be defined with a `:revision` tag. Do not forget the “RUN=YES” attribute if you want your document to have the change-bar inserted in the processed document.

Compare Types: LINE and WORD



REFID=*name* Name of the revision identifier for the IBM BookMaster :rev/:erev. tags.

RCVAL=*number* Numeric revision code for SCRIPT/VS revision tags.

Example	Description
`REVREF  REFID=ABC`	IBM BookMaster example `:rev refid=ABC.` and `:erev refid=ABC.` tags.
`REVREF  RCVAL=5`	SCRIPT/VS example `.rc 5` on/off delimiters.

Search Strings in the Input File

There are two process options to search for strings in the input file:

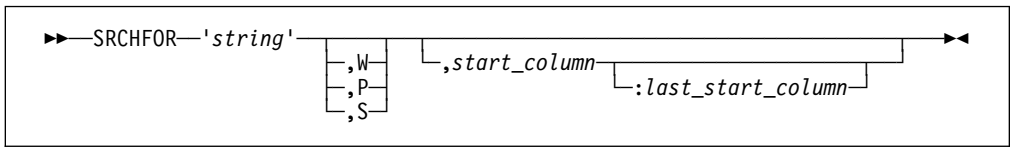
SRCHFOR Search a text string in the input file

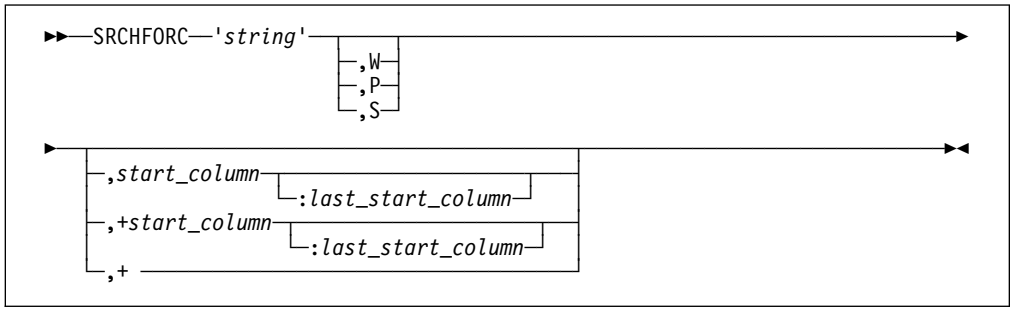
SRCHFORC Search a text string continuation

These statements search for a specified string in the input Search file. The string may be further qualified as a word, prefix, or suffix, and where it must be positioned on the line.

SRCHFORC is the continuation of the immediately preceding SRCHFOR or SRCHFORC process statement. In the case of a SRCHFOR/SRCHFORC group, all of the specified strings must occur on the same line for the search to be successful.

Compare Type: Search





string The character or hexadecimal string to be searched for (enclosed by single quotes). Use two consecutive single quotes ('') for quotes within the search string.

W Word. String must appear as a separate *word*. That is, be delimited by one or more spaces or special characters.

P Prefix. String must appear as the first part of some other text.

S Suffix. String must appear as the last part of some other text.

start_column The column in which the string must start for the search to be successful. (If an *last_start_column* is also specified, see description for that operand.)

last_start_column The "latest" column in which the string can start for the search to be successful. (Must be separated from the *start_column* by a colon.)

+start_column The relative column (starting from the column where the string for the previous SRCHFOR/SRCHFORC was found) in which the string must start for the search to be successful. (A corresponding *last_start_column* operand can be specified in a similar way to that for the *start_column*.)

+ The string specified can occur anywhere after the position of the previously found string for the search to be successful.

Example	Description
`SRCHFOR 'ABC'`	Searches for string “ABC”
`SRCHFOR 'ABC',W`	Searches for the word “ABC”
`SRCHFOR X'4004'`	Searches for the hexadecimal string X'4004'
`SRCHFOR 'A''bc'`	Searches for string “A'bc”
`SRCHFOR 'ABC',5:10`	Searches for string “ABC” starting in positions 5 to 10
`SRCHFOR 'ABC',W,5`	Searches for the word “ABC” starting in position 5
`SRCHFOR 'ABC'` `SRCHFORC 'DEF'`	Searches for strings “ABC” and “DEF” in any order in the same line.
`SRCHFOR 'ABC'` `SRCHFORC 'DEF',+`	Searches for the string “DEF” following the string “ABC”
`SRCHFOR 'ABC'` `SRCHFORC 'DEF',W,+`	Searches for the word “DEF” following the string “ABC”
`SRCHFOR 'ABC'` `SRCHFORC 'DEF',+5`	Searches for the string “DEF” in the 5th position after the string “ABC”

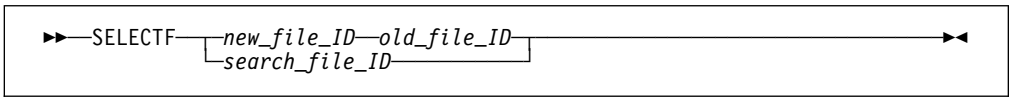
`SRCHFOR 'ABC'`
`SRCHFORC 'DEF',+5`
`SRCHFORC 'GKL'`
	Searches for the string “DEF” in the 5th position after the string “ABC” with the string “GKL” also anywhere in the same line

Select Files from a List of Files (CMS)

The SELECTF process statement (for CMS) selects file pairs to be compared or a single file to be searched. The SELECTF process statement overrides the source file names from the Primary Compare Menu or the names specified on the CMS command line.

A single SELECTF process statement may contain a “wildcard” character (“*”) anywhere in the *filename* or *filetype*. (For *new* and *old* files, the “*” must be used in exactly the same way.) Multiple SELECTF process statements may have an “*” only as the *filemode* (*fm*) part of the file ID.

Compare Types: FILE, LINE, WORD, BYTE, and Search



new_file_ID Fully qualified *new* file ID: fn ft fm.

old_file_ID Fully qualified *old* file ID: fn ft fm.

search_file_ID Fully qualified search file ID: fn ft fm.

Example	Description
`SELECTF NEW1 TEST A OLD1 TEST A`	Selects files NEW1 TEST A and OLD1 TEST A for comparison with each other.
`SELECTF NEW1 TEST * OLD1 TEST *`	Selects the group of files NEW1 TEST (all file modes) for comparison with the group of files OLD1 TEST (all file modes). Files compared according CMS order.
`SELECTF NEW* TEST A OLD* TEST A`	Selects the group of files with file names beginning with “NEW” (and file type TEST and file mode A) for comparison with the group of files with file names beginning with “OLD” (and file type TEST and file mode A). (Valid for single SELECTF statement only.)
`SELECTF NEW* TEST A OLD TEST* A`	Invalid use of “wildcard” character. (“*” must be used in exactly the same way for both files.)
`SELECTF NEW1 TEST* A`	(Example of SELECTF being used for search.) Selects the group of files with file name NEW1, file type beginning with “TEST” and file mode of A.

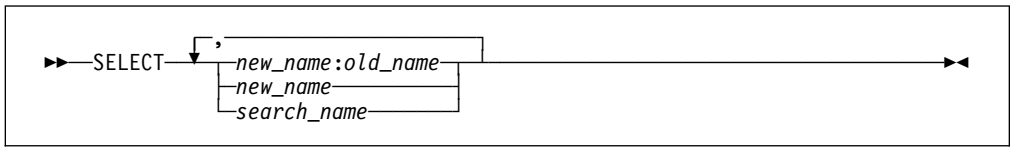
Select Members or Files (CMS)

The SELECT process statement (for CMS) selects members from a macro library (MACLIB) or a text library (TXTLIB), or selects files in conjunction with a file ID specified as “* *ft fm*” for comparison or for being searched. You can specify as many member/file names as fit on one line. If you need to select additional members/files, enter a new SELECT statement.

For comparisons, the new members/files are normally compared with old members/files that have the same names. Use the colon character (:) to compare members/files that are not named alike.

Any number of SELECT statements may be specified.

Compare Types: FILE, LINE, WORD, BYTE, and Search



new_name The name of a new member/file that is to be compared to an old member/file.

old_name The name of an old member/file that does not have a like-named member/file in the new MACLIB/TXTLIB or file group. This member/file name, if entered, must be separated from the *new_name* name by a colon (:).

If the *old_name* name is not used, SuperC attempts to compare the *new_name* to a like-named member/file of the old MACLIB/TXTLIB or file group.

search_name The name of the member/file that is to be searched.

Example	Description
`SELECT  NEW1,NEW2`	For a MACLIB/TXTLIB: For a comparison, compares member NEW1 from the *new* MACLIB/TXTLIB with the member NEW1 from the *old* MACLIB/TXTLIB and compares member NEW2 from the *new* MACLIB/TXTLIB with the member NEW2 from the *old* MACLIB/TXTLIB. For a search, selects members NEW1 and NEW2 from the MACLIB/TXTLIB to be searched. For a "* *ft fm*" file group: For a comparison, compares filename NEW1 from the *new* file group with the filename NEW1 from the *old* file group and compares filename NEW2 from the *new* file group with the filename NEW2 from the *old* file group. For a search, selects filenames NEW1 and NEW2 from the file group to be searched.

Select Members (VSE)

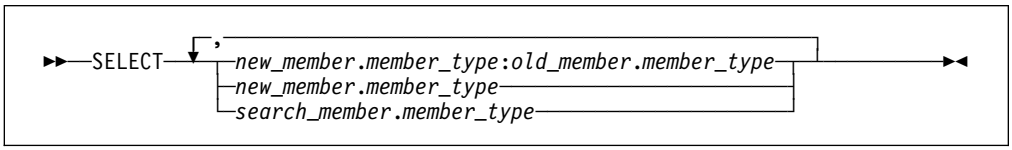
The SELECT process statement (for VSE) selects members from a sublibrary of a Librarian library for comparison or for being searched. You can specify as many member names as fit on one line. If you need to select additional members, enter a new SELECT statement.

Note: The names of the Librarian library and sublibrary (from which the members are to be selected) must be specified using NEWDD and OLDDD process statements (see "DD-VSE DLBL/TLBL Definitions" on page 246).

For comparisons, the new members are normally compared with old members that have the same names. Use the colon character (:) to compare members that are not named alike.

Any number of SELECT statements may be specified.

Compare Types: FILE, LINE, WORD, BYTE, and Search



new_member.member_type

The name of a member and its member type in the sublibrary of the Librarian library in the *new* file that is to be compared to a member in the *old* file.

old_member.member_type

The name of a member and its member type in the sublibrary of the Librarian library in the *old* file that does not have a like-named member in the *new* file. This member name and member type, if entered, must be separated from the *new_member* name and member type by a colon (:).

If the *old_member* name is not used, SuperC attempts to compare the *new_member* to a like-named member in the sublibrary of the Librarian library in the new file.

search_member.member_type
The name of a member and its member type in the sublibrary of the Librarian library that is to be searched.

Example	Description
`SELECT NEW1.C,NEW2.C,NEW3.C`	For a comparison, compares member NEW1.C from the sublibrary of the Librarian library of the *new* file with the member NEW1.C from the sublibrary of the Librarian library of the *old* file, compares member NEW2.C from the sublibrary of the Librarian library of the *new* file with the member NEW2.C from the sublibrary of the Librarian library of the *old* file and compares member NEW3.C from the sublibrary of the Librarian library of the *new* file with the member NEW3.C from the sublibrary of the Librarian library of the *old* file. For a search, selects members NEW1.C, NEW2.C, and NEW3.C from the sublibrary of the Librarian library to be searched.
`SELECT NEW1.C:OLD1.C,MEMBER2.C`	Compares member NEW1.C from the sublibrary of the Librarian library of the *new* file with the member OLD1.C from the sublibrary of the Librarian library of the *old* file and compares member MEMBER2.C from the sublibrary of the Librarian library of the *new* file with the member MEMBER2.C from the sublibrary of the Librarian library of the *old* file.

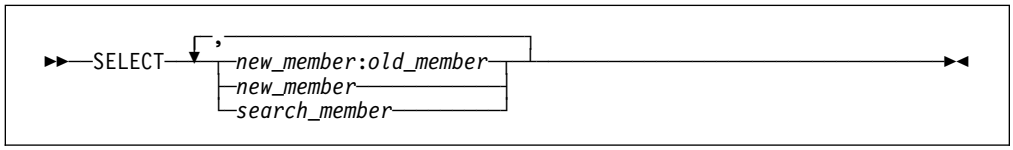
Select PDS Members (MVS)

The SELECT process statement (for MVS) selects members from a PDS for comparison or for being searched. You can specify as many member names as fit on one line. If you need to select additional members, enter a new SELECT statement.

For comparisons, the new members are normally compared with old members that have the same names. Use the colon character (:) to compare members that are not named alike.

Any number of SELECT statements may be specified.

Compare Types: FILE, LINE, WORD, BYTE, and Search



new_member The name of a new PDS member that is to be compared to an old PDS member.

old_member The name of an old PDS member that does not have a like-named member in the new PDS. This member name, if entered, must be separated from the *new_member* name by a colon (:).

If the *old_member* name is not used, SuperC attempts to compare the *new_member* to a like-named member of the old PDS.

search_member The name of the PDS member that is to be searched.

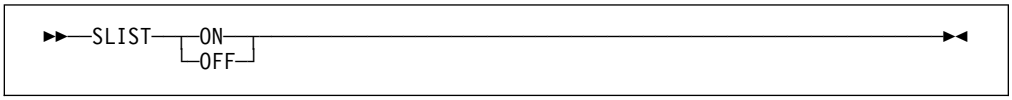
Example	Description
`SELECT NEW1,NEW2`	For a comparison, compares member NEW1 from the *new* PDS with the member NEW1 from the *old* PDS and compares member NEW2 from the *new* PDS with the member NEW2 from the *old* PDS. For a search, selects members NEW1 and NEW2 from the PDS to be searched.
`SELECT NEW1:OLD1,MEMBER2`	Compares member NEW1 from the *new* PDS with the member OLD1 from the *old* PDS and compares member MEMBER2 from the *new* PDS with the member MEMBER2 from the *old* PDS.

Statements File Listing Control

The SLIST process statement turns the printing of process statements in the output listing on and off.

The initial setting of this control is ON.

Compare Types: FILE, LINE, WORD, BYTE, and Search



ON Causes the lines in the process statements file following the SLIST statement to be listed in the output listing.

OFF Causes the lines in the process statements file following the SLIST statement to be suppressed in the output listing.

Example	Description
`SLIST OFF`	Do not list following process statements.
`SLIST ON`	List following process statements.

Title Alternative Listing

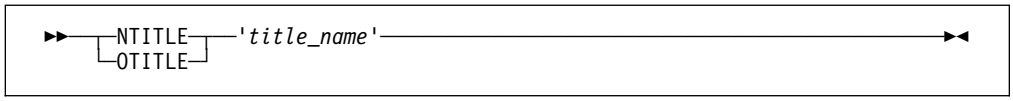
There are two process statements that let you provide an alternative title:

NTITLE New (or search) listing file title identification

OTITLE Old listing file title identification

These statements allow an alternative file identification to be used in the output listing (instead of the default identifiers “New File ID” and “Old File ID”).

Compare Types: FILE, LINE, WORD, BYTE, and Search (NTITLE only)



title_name The alternative title to be used on the output listing to identify either the “new” file (NTITLE) or the “old” file (OTITLE). The title name must be in single quotes and may be up to 54 characters in length. Use two consecutive quotes for quotes within the title name.

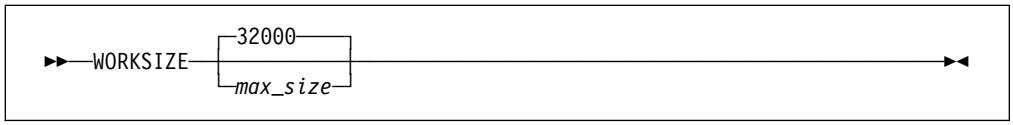
Example	Description
`NTITLE 'New Title'`	Change title heading for new (or search) file to “NEW TITLE”
`OTITLE 'Old Title'`	Change title heading for old file to “OLD TITLE”

Work Size

The WORKSIZE process statement allows the maximum size of the comparison set to be adjusted for comparing large files.

If WORKSIZE exceeds 99999, then the SuperC LINE comparison DELTA listing type may result in wider columns for LEN N-LN# and O-LN#. Typically, these columns contain 5-digit values. However, when WORKSIZE exceeds 5 digits, and providing the standard record length of the listing is not affected, the columns are extended to contain 7-digit values. If the length of the input source lines in the listing are such that 7-digit values cannot fit, the report outputs 5-digit values by default, and only reports 7-digit values when significant characters would otherwise be lost.

Compare Type: FILE, LINE, WORD, BYTE. It is ignored if specified on a SEARCH.



max_size The maximum number of units for comparison. Maximum value is 9999999.

Year Aging

There are two process statements for year aging:

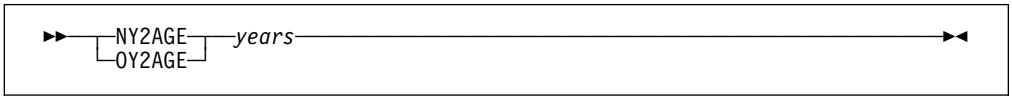
NY2AGE Aging applies to the new file

OY2AGE Aging applies to the old file

These statements age all of the defined dates in either the new or old file. That is, the number of years specified is added to the "year" portion of each defined date in the file concerned.

Note: Dates are *defined* by the Date Definition process statements NY2C, NY2Z, NY2D, NY2P, OY2C, OY2Z, OY2D and OY2P; see "Date Definitions."

Compare Type: LINE



years A number (0 to 999) indicating the number of years by which all defined dates in the file are to be aged.

Example	Description
`OY2AGE 28`	Ages all defined dates in the "old" file by 28 years before being compared. The listing shows the original date. For example, a defined date in the "old" file with a value equating to March 1, 1997, is aged to March 1, 2025 before being compared to its equivalent in the "new" file.

Date Definitions

There are eight process statements that set date definitions:

NY2C New file, date in character format
NY2Z New file, date in zoned decimal format
NY2D New file, date in unsigned packed decimal format
NY2P New file, date in packed decimal format
OY2C Old file, date in character format
OY2Z Old file, date in zoned decimal format
OY2D Old file, date in unsigned packed decimal format
OY2P Old file, date in packed decimal format

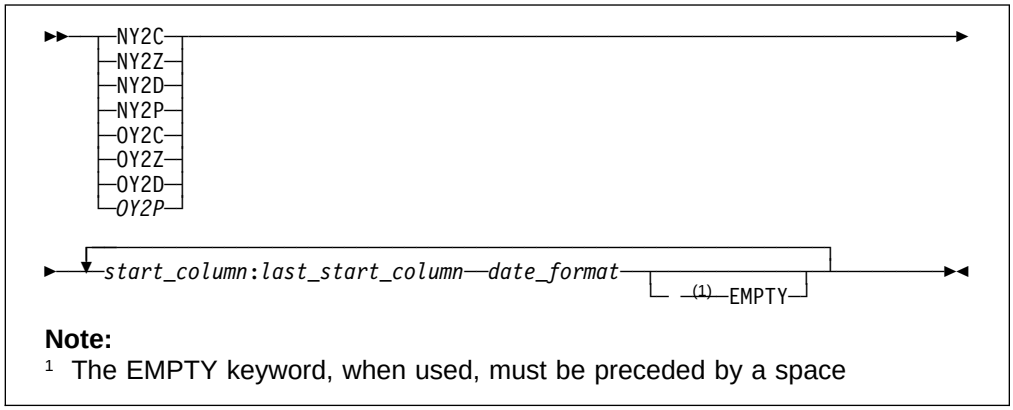
Notes:

1. If any Date Definition process statements are used, a Y2PAST process statement should also be used so that the "century" portion of the date can be determined where necessary. (If a Y2PAST process statement is not present, a default fixed window based on the current year is used.)
2. For a description of each date format (character, zone, decimal and packed), see "Date Formats (Keyword suffixes: C, Z, D, P)" on page 265.
3. If any Date Definition process statements are used, an *information line* is generated on the listing output (see Figure 80 on page 276).
4. Do not use any Date Definition process statements if using the COLHEAD process statement.

Defines the location and format of a date field on the input file. Up to 254 date definition statements can be entered for each file. The matching of the new to the old dates is performed according to the sequence that the statements are entered. That is, the first defined *old* date is matched to the first defined *new* date.

If the number of date definition statements for one file differ from the number of date definition statements for the other file, the location and format details for the "missing" date definition statements are assumed to be the same as their counterpart date definition statements for the other file.

Compare Type: LINE



Note:
[1] The EMPTY keyword, when used, must be preceded by a space

start_column The first position of the date in the input file.

last_start_column The last position of the date in the input file. (Must be separated from the *start_column* by a colon.)

date_format A mask representing the format of the date.

For a Julian date, the mask must be either YYDDD or YYYYDDD.

For date formats other than Julian, the mask must contain 2 "D"s (representing the day part of the date field), 2 "M"s (representing the month), and either 2 or 4 "Y"s (representing the year) or, if the date contains a year only, it must contain either 2 or 4 "Y"s.

If the date is character, there may also be a separator between the different parts. In this case, you can represent the position of the separators by one of the following:

- S (indicates that this position within the date is not used in comparison)
- . (period, used in comparison)
- / (forward slash, used in comparison)
- : (colon, used in comparison)

Note: The length of the *date_format* mask must correspond to the length of the date in the input file as indicated by the values of *start_column* and *last_start_column*.

EMPTY This keyword is optional. If it is entered, the defined date field is checked for containing zeros, spaces, low-values, or high-values before commencing the comparison process. If

any of these values are found, the date is not converted according to the Y2PAST criteria but instead is converted to an extended format with the initial value. For example, a date defined by the process statement `OY2C YYMMDD` which contains all zeros is compared as “YYYYMMDD” with a value of zeros.

Example	Description
`NY2C 1:8 MMDDYYYY 9:16 MMDDYYYY 21:28 YYYYMMDD`	The new file has dates in character format in columns 1 to 8, 9 to 16 and 21 to 28.
`OY2P 5:8 YYMMDD 9:12 YYMMDD`	The old file has dates in packed decimal format in columns 5 to 8 and 9 to 12.
`OY2P 101:104 MMDDYY`	The old file has a date in packed decimal format in columns 101 to 104,
`NY2Z 101:108 YYYYMMDD`	The new file has a date in zoned decimal format in columns 101 to 108.
`NY2C 101:110 YYYY.MM.DD`	The new file has a date in character format (with separators) in columns 101 to 110.
`OY2C 93:98 DDMMYY EMPTY`	The old file has a date in character format in columns 93 to 98. If the date field contains zeros, spaces, low-values, or high-values, the date in the old file is converted before being compared to an extended format (DDMMYYYY) with a value of all zeros, spaces, low-values, or high-values respectively.

Date Formats (Keyword suffixes: C, Z, D, P)

C Character date data.

Examples:

'96' is represented as hexadecimal X'F9F6'

If using a MMDDYY format, March 21, 1996 is represented as hexadecimal X'F0F3F2F1F9F6'

Z Zoned decimal date data. The date can be represented as follows:

X'xyxy' to X'xyxyxyxyxyxyxy'

y is hexadecimal 0 to 9 and represents a date digit.
x is hexadecimal 0 to F and is ignored.

Examples:

'96' is represented as hexadecimal X'F9C6' or X'0906'

'03211996' is represented as hexadecimal X'F0F3F2F1F1F9F9C6' or X'0003020101090906'

P Packed decimal date data. The date can be represented as follows:

X'zyyx' to X'zyyyyyyyyx'

y is hexadecimal 0 to 9 and represents a date digit.
x is hexadecimal A to F and is ignored.
The z part is normally zero but is not ignored.

Examples:

'96' is represented as hexadecimal X'z96F' or X'z96C'

'1996' is represented as hexadecimal X'z1996C'

'03211996' is represented as hexadecimal X'z03211996x' (the x part is ignored).

'96203' (a Julian date) is represented as hexadecimal X'96203C'

D Unsigned packed decimal date data. The date can be represented as follows:

'yy' to 'yyyyyyyy'

y is hexadecimal 0 to 9 and represents a date digit.

Examples:

'96' is represented as hexadecimal X'96'

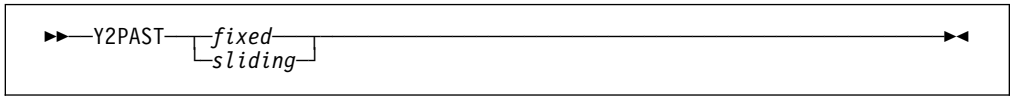
'03211996' is represented as hexadecimal X'03211996'

Global Date

The Y2PAST process statement specifies a 100-year period (used for determining the century-part of a date when only a 2-digit year has been specified). The Y2PAST process statement uses either a fixed or sliding window.

Note: The Y2PAST process statement should always be used if any of the Date Definition process statements (NY2C, NY2Z, NY2D, NY2P, OY2C, OY2Z, OY2D, OY2P) have also been used.

Compare Type: LINE



fixed A 4-digit number indicating a fixed window.
sliding A 1-digit or 2-digit number indicating a sliding window.

Example	Description
`Y2PAST 1986`	A fixed window specifying a 100-year period from 1986 to 2085.
`Y2PAST 70`	A sliding window specifying (based on the current year being 2001) a 100-year period from 1931 (70 years in the past) to 2030.
`Y2PAST 5`	A sliding window specifying (based on the current year being 2001) a 100-year period from 1996 (5 years in the past) to 2095.

CMS Command Line Option Directives

Command option directives are options that the SuperC EXEC intercepts and interprets. They are not passed to the SuperC program as parameters like process options.

ERASRC0 Erase the listing file if the return code from the SuperC program is zero (that is: for a compare, the files were the same; for a search, no matches were found).

Note: If you do not specify the ERASRC0 command line option directive, a listing file is generated (unless you have used the NOLIST listing type) even when the return code is zero.

MENU Display the Primary Comparison Menu after accepting the input parameter list. The options on the CMS command line are verified and put into the proper fields of the Primary Comparison Menu. This allows you to use SuperC menu mode from FILELIST and also uses the Options List file.

Notes:

1. If you do not specify the MENU command line option directive, SuperC performs the comparison immediately you press Enter.
2. The MENU line command option directive does not apply to the SuperC Search.

NOIMSG No information messages. Do not generate information messages. Warning and error messages should still be displayed.

NONAMES The SUPERC NAMES * file is not to be used in determining the options to be sent to the SuperC Comparison.

Notes:

1. The NONAMES line command option directive does not apply to the SuperC Search.
2. Refer to “Command Line Priority and Overriding” on page 198.

NOOLF The default-named Options List file is not to be used in determining the options to be sent to SuperC.

Notes:

1. For comparisons, the default name for the Options List file is SUPERC OLIST A
2. For searches, the default name for the Options List file is SRCHFOR OLIST A
3. NOOLF only suppresses the use of the default-named Options List file. If you have entered the OLF keyword on the CMS command line (to specify your own-named Options List file) and you have also entered NOOLF, any options contained in your own-named Options List file are still used.

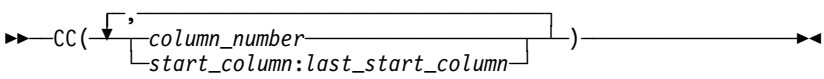
PRINT Print the comparison results.

CMS Command Line Statement Option Directives

Command line statement option directives are options that are interpreted and transformed into SuperC process statements. (The newly-created process statements are passed to SuperC in the temporary control file, SUPERC $SYSIN$ A.)

For a full description of each process statement, see “Process Statements” on page 236.

CC Compare Columns. This is the command line directive for the Compare Columns (CMPCOLM) process statement. It allows you to select specified columns of data (by entering either single column numbers or ranges of column numbers) to be compared from each file or to be searched from the search file. Up to 15 separate single column numbers and column ranges can be specified.



Note:
[1] Separator can be a comma or a space.

column_number The single-column to be compared or searched.

start_column The starting column to be compared or searched.

last_start_column The ending column to be compared or searched. (Must be separated from the *start_column* by a colon.)

Example	Description
`ASMFSUPC...(CC(2:75)...`	Compare columns 2 to 75 in both files (or search those columns in the search file).
`ASMFSUPC...(CC(1:10,25:45,75)...`	Compare columns 1 to 10, 25 to 45, and column 75 in both files (or search those columns in the search file).
`ASMFSUPC...(CC(1:10 25:45 75)...`	Same as previous example.

Valid for FILE, LINE, WORD, BYTE compare types and Search.

LC List Columns. This is the command line directive for the List Columns (LSTCOLM) process statement. It allows you to selectively list a range of columns to be listed in the listing output file. Only one column range may be specified (enclosed within parentheses). The column range is denoted by the first column number, a colon (:), and the last column number (with no spaces on either side of the colon).

Example	Description
`ASMFSUPC...(LC(1:45)...`	List columns 1 to 45.

Valid for FILE, LINE, WORD, BYTE compare types and Search.

LT Line Count. This is the command line directive for the LNCT process statement. LT indicates how many lines of output should appear on each page. The line count value (enclosed within parentheses) must range between 15 and 999999.

Example	Description
`ASMFSUPC...(LT(55)...`	List 55 lines per page.

Valid for FILE, LINE, WORD, BYTE compare types and Search.

RR Revision Code Reference. This is the command line directive for the REVREF process statement and is used in conjunction with the UPDREV process option. Use REFID to indicate :rev. and :erev. tags for IBM BookMaster documents and .rc for other SCRIPT/VS documents. In either case, the details are enclosed within parentheses following the RR command line statement option directive.

Example	Description
`ASMFSUPC...(UPDREV RR(REFID=LV12)`	IBM BookMaster revision reference identifier.
`ASMFSUPC...(UPDREV RR(RCVAL=1)`	SCRIPT/VS (not IBM BookMaster) revision code.

Valid for LINE and WORD compare types. (Not Search.)

Understanding the Listings

SuperC allows you to produce a range of listings (reports) which provide detailed information on the results of your comparison or search.

General Listing Format

The format and content of each type of listing depends on:

- Whether you are using the SuperC Comparison or the SuperC Search
- The *listing type* used (see “Listing Type” on page 186)

  Note: The NOLIST listing type suppresses the generation of any listing output or listing file.
- Whether you are comparing (or searching) a *single* file or a file *group*
- The *compare type* used (in the case of the SuperC Comparison)
- The *process options* used
- The *process statements* used

Note: Dates in the heading lines on the sample listing output in this document appear in the format MM/DD/YYYY. This is the date format for VSE and CMS listings. The dates in the heading lines for MVS outputs appear in the format YYYY/MM/DD.

How to View the Listing Output

The listing output is always written to a *listing file* (unless the NOLIST listing type is used), from which you can subsequently print the listings.

For details on the *naming* of the listing file:

- On MVS, see “Invoking the Comparison on MVS” on page 180.
- On CMS, see “Listing File ID” on page 187.
- On VSE, see “Invoking the Comparison on VSE” on page 199.

On CMS, the listing output is also normally displayed on the screen immediately after the comparison or search process has finished.

The following pages contain:

- A description of the general format of the *comparison listing* (page 270), followed by examples of various listings produced by the SuperC Comparison.
- A description of the general format of the *search listing* (page 283), followed by examples of various listings produced by the SuperC Search.

Notes:

1. Some of the sample listings have been edited to fit on a page. An "|...|" shows text has been removed.
2. The sample listings shown are for CMS. Most cases show, "CMS" in the page heading and CMS-style file IDs (fn ft fm).

   However, the format and content of the listing output is almost identical, regardless of which platform you are using to run SuperC. The only significant differences are:

   On MVS
   - "MVS" is shown in the page heading
   - PDS member names are shown
   - In the case of file *groups*, PDS names are shown

   On VSE
   - "VSE" is shown in the page heading
   - Librarian library, sublibrary, member names, and member types are shown
   - In the case of file *groups*, Librarian library, and sublibrary names are shown

The Comparison Listing

SuperC comparison listings consist of 4 basic parts (although not all parts are present for all types of listing output produced):

- Page Headings (see below)
- Listing Output Section (see page 271)
- Member Summary Listing (CMS) (see page 273)
- Overall Summary Section (see page 275)

Page Headings

SuperC generates a page heading at the top of each page. The heading consists of two lines of information.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  12.31   PAGE    1
  NEW:  NEW      TEST1    A              OLD:  OLD      TEST1    A
```

Figure 76. Example of Page Heading Lines for the Comparison Listing

Figure 76 shows typical page heading lines. The first line contains:

- Printer control page eject character ("1" in column one. Not present when the NOPRTCC process option is specified)
- "Platform-identifier" This is one of "CMS," "MVS," or "VSE."
- Program identification title including version and the version date: V1R5M0 (06/11/2004)
- The date and time of the compare

- The page number

Note: The program version and program date are important when reporting suspected SuperC problems.

The second heading line identifies the *new* and *old* files. Normally this line shows the file IDs of the *new* and *old* files. However, if the NTITLE and OTITLE process statements have been specified then the corresponding alternative file titles are shown instead of the file IDs.

Listing Output Section

The listing output section shows where and what the changes are. Figure 77 is an example of a Listing Output Section for a LINE comparison with a listing type of DELTA.

```
1              LISTING OUTPUT SECTION (LINE COMPARE)

2  ID      SOURCE LINES           |...|        TYPE  LEN N-LN# O-LN#
3      ----+----1----+----2----+-|...|-+----8
4  I - 970521    ▒▒▒▒▒▒           |...|        RPL=    1 00001 00001
5  INFO  Date cols 11:15 packed 2|...|
6  D - 970522    ▒▒▒▒▒
5  INFO  Date cols 11:14 packed 1|...|
```

Figure 77. Example of the Listing Output Section of the Comparison Listing

1 Section title line. It tells you that this is a LINE comparison. Possible compare types are BYTE, FILE, LINE, and WORD.

2 Column header line.

ID A two-column prefix code that identifies the status of the line. See “Listing Prefix Codes” on page 272.

SOURCE LINES The actual text or data from the source files. Under this heading, the actual data from the files is listed.

TYPE Further breakdown of the ID field. See “Type of Difference Codes” on page 273 for information on TYPE codes.

LEN The “length” or number of consecutive lines of the selected type.

N-LN# Indicates the relative record (line) number of this line (or where it is to be inserted) in the *new* source file. Numbers are in decimal.

O-LN# Indicates the relative record (line) number of this line (or where it was deleted from) in the *old* source file. Numbers are in decimal.

3 The scale of the column positions of the input source lines.

4 An inserted (I) line. The RPL in TYPE indicates that it is a replacement line. This replacement involves the line 00001 in both files.

Note: Occasionally, you may see some “unusual” characters on the inserted (I) and deleted (D) lines. These characters simply represent data that is in a non-character (and therefore not

directly printable) format in the input record and should be ignored.

5 An information line that is generated on a comparison listing when a Date Definition process statement is used (see “Date Definitions” on page 263) and when the preceding inserted (I) line or deleted (D) line contains a date. The information line shows you the content of the date field as it exists on the input file and the date as used in the comparison. For a full example, see Figure 80 on page 276.

6 A deleted (D) line.

Listing Prefix Codes: SuperC output lines are flagged with the following prefix codes listed under the ID column:

(space) *Match* No prefix code means the data is the same in both files.

I *Insert* Data that is in the *new* file, but is missing[2] from the sequence in the *old* file.

D *Delete* Data that is in the *old* file, but is missing[2] from the sequence in the *new* file.

DR *Delete Replace* For BYTE compare type only. The bytes in the *old* file that were replaced by the bytes shown in the preceding insert (I) line.

RN *Reformat New* For LINE compare type only. A reformatted line in the *new* file. This line contains the same data as the *old* file line, but with different spacing.

RO *Reformat Old* For LINE compare type only. A line in the *old* file that is reformatted in the *new* file. This line is not shown if the DLREFM process option is used.

MC *Match Compose* For WORD compare type only. A line containing words that match. The line may also contain spaces to show the relationship between the matching words and any inserted or deleted words. Inserted and deleted words are shown in following insert compose (IC) and delete compose (DC) lines. See Figure 87 on page 283 for an example using a WORD compare type.

IC *Insert Compose* For WORD compare type only. A line containing words from the *new* file that are not in the *old* file. This line usually follows a match compose (MC) line.

DC *Delete Compose* For WORD compare type only. A line containing words from the *old* file that are not in the *new* file. This line usually follows a match compose (MC) or insert compose (IC) line.

IM *Insert Moved* For comparison listings created using the FMVLNS (flag moved lines) process option. A line in the *new* file that also appears in the *old* file, but has been moved. If the line was reformatted, this is indicated by a flag to the right of the listing.

DM *Delete Moved* For comparison listings created using the FMVLNS (flag moved lines) process option. A line in the *old* file that also appears in the *new* file, but has been moved. If the line was reformatted, this is indicated by a flag to the right of the listing.

---

[2] “Missing” data is data that is missing from the data sequence but may exist in some other part of the file.

| *Change Bar* For comparison listings created using the GWCBL (generate WORD/LINE comparison change bar listing) process option. A change bar showing that words/lines were either inserted or deleted.

Type of Difference Codes: At the far right of some listings are headings that provide additional information about the numbers and types of differences SuperC has found. Headings you may see are:

MAT= Number of matched lines.

RFM= Number of reformatted lines.

RPL= Number of replaced lines.

INS= Number of lines that are in the *new* file, but missing in the *old* file.

DEL= Number of lines that are in the *old* file, but missing in the *new* file.

IMR= Number of lines in the *new* file that have been moved from where they were in the *old* file and reformatted. Listing should show a matching "DMR=" flag for a line in the *old* file.

DMR= Number of lines in the *old* file that have been moved and reformatted in the *new* file. Listing should show a matching "IMR=" flag for a line in the *new* file.

IMV= Number of lines in the *new* file that have been moved from where they were in the *old* file. Listing should show a matching "DMV=" flag for a line in the *old* file.

DMV= Number of lines in the *old* file that have been moved in the *new* file. Listing should show a matching "IMV=" flag for a line in the *new* file.

Member Summary Section (CMS)

SuperC generates the member summary section when you specify either a file group comparison or use a macro library (MACLIB) or text library (TXTLIB) as input. The member summary section is really two sections with a page separator between them.

Figure 78 on page 274 shows an example of the two member summary sections for a FILE compare type.

The first section indicates which files were compared and whether they were found to be different or the same. In Figure 78 on page 274, NEW TEST1 A was compared to OLD TEST1 A and NEW TEST2 A was compared to OLD TEST2 A. Both comparisons found differences. Following the member statistics are the group statistics. As this was a FILE comparison, the statistics are in terms of files and the number of bytes in each file.

Note: Different compare types produce slightly different results in the first section.

The second part of the member summary section shows all of the members from both the *new* and *old* file groups which were not paired (and hence not compared). In Figure 78 on page 274, only OLD TEST3 A from the *old* file group was not compared to any file from the *new* group.

```
1                        MEMBER SUMMARY LISTING (FILE COMPARE)

2  DIFF SAME              FILE NAMES                N-BYTES O-BYTES

3   **       NEW     TEST1    A5 OLD    TEST1    A5     240     240
    **       NEW     TEST2    A5 OLD    TEST2    A5     240     240
                    ----------------------          ------- -------
4                         GROUP TOTALS                  480     480


5       2   TOTAL FILES PROCESSED AS A GROUP
6       2   TOTAL FILES PROCESSED HAD CHANGES
7       0   TOTAL FILES PROCESSED HAD NO CHANGES
8       0   TOTAL NEW FILES NOT PAIRED
9       1   TOTAL OLD FILES NOT PAIRED

1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
 NEW:  NEW      TEST*    A                                    OLD:

                      MEMBER SUMMARY LISTING (FILE COMPARE)

        NON-PAIRED NEW GROUP FILES           |     NON-PAIRED OLD GR

                                             | 10  OLD   TEST3  A5
```

Figure 78. Example of the Member Summary Section of the Comparison Listing

1 Section Header. In this context, “member” can refer to either members of a MACLIB or TXTLIB, or members of a file group.

2 Header line. Consists of several column headers.

DIFF Contains “**” when the *new* and *old* files differ.

SAME Contains “**” when the *new* and *old* are the same.

FILE NAMES The file names or paired members of the file group or MACLIB/TXTLIB compared.

N-BYTES Number of bytes processed in the *new* member.

O-BYTES Number of bytes processed in the *old* member.

N-LINES (Not shown) Number of lines processed in the *new* member.

O-LINES (Not shown) Number of lines processed in the *old* member.

N-HASH-SUM (Not shown) SuperC generated a hash value for the *new* member.

O-HASH-SUM (Not shown) SuperC generated a hash value for the *old* member.

Note: The hashsums of files can be used to compare two files that are not physically on the same system. If the hashsum of a file on system A is different from the hashsum of a file on system B, then the files can be said to be different. If the hashsum of the files are identical, there is a high probability that the files are the same. You should also compare

the number of lines and number of bytes as secondary confirmation that the files are the same.

3 Group (file) file statistics.

4 Group totals header line.

5 Total number of files that were processed as a group.

6 Total number of files compared that had differences.

7 Total number of files compared that had no differences.

8 Total number of *new* files that were not paired (and therefore were not compared).

9 Total number of *old* files that were not paired (and therefore were not compared).

10 OLD TEST3 A5 was present in the *old* file group. It could not be paired with a similarly named file in the *new* file group and was not processed.

Overall Summary Section

The overall summary section gives the overall statistics of the comparison process. Figure 79 is a representative example of an overall summary section.

```
1         LINE COMPARE SUMMARY AND STATISTICS

2  2 NUMBER OF LINE MATCHES     8  1  TOTAL CHANGES (PAIRED+NONPAIR
3  0 REFORMATTED LINES          9  1  PAIRED CHANGES (REFM+PAIRED
4  1 NEW FILE LINE INSERTIONS  10  0  NON-PAIRED INSERTS
5  1 OLD FILE LINE DELETIONS   11  0  NON-PAIRED DELETES
6  3 NEW FILE LINES PROCESSED
7  3 OLD FILE LINES PROCESSED

12  LISTING-TYPE = OVSUM   13  COMPARE-COLUMNS =    1:72
14  LONGEST-LINE = 80
15  PROCESS OPTIONS USED: NONE
```

Figure 79. Example of the Overall Summary Section of the Comparison Listing

Figure 79 shows the following information about the comparison:

1 The first word of the title tells you the type of comparison. The overall summary is provided for BYTE, FILE, LINE, and WORD compare types.

2 Of the 3 lines in each file, 2 from the *new* file matched 2 corresponding lines of the *old* file. These are called matching lines.

3 There are no reformatted lines.

4 There is 1 inserted line in the *new* file.

5 The *old* file contains 1 line that is missing from the *new* source file.

6 3 lines from the *new* file were processed.

7 The *old* file also has a total of 3 lines.

8 The total number of changes is a summation of items **9**, **10**, and **11**. It is a convenient number that best represents the change activity of the two compared files.

9 The total number of reformats and paired changes. This represents a sum of items that may be considered to be a single change. That is, some changes are made in pairs and need only be counted as a single instance of a change.

10 There were no non-paired inserts. Non-paired inserts are changes to the *new* file that have no relationship to the *old* file (that is, no deletes from the *old* file occurred in the same area).

11 There were no non-paired deletes. Non-paired deletes are changes to the *old* file that have no relationship to the *new* file (that is, no inserts to the *new* file occurred in the same area).

12 The listing type is OVSUM. This is the listing type option selected for the comparison. Other options are: DELTA, CHNG, and LONG.

13 SuperC compared columns 1 through 72. This value provides a convenient reference for confirming if all of the columns in the line have been compared or only some portion of the line.

14 The longest line length of any line in either file is 80 characters.

15 No process options were used.

Examples of Comparison Listings

The following represent some of the output types available from SuperC.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  12.31   PAGE    1

NEW:  D1       A        A                                    OLD:  D2       A        A

                      LISTING OUTPUT SECTION (LINE COMPARE)

ID      SOURCE LINES                                                                    TYPE  LEN N-LN# O-LN#
    ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
                                                                                        MAT=    1
I - 970522     Õ†–¢                                                                     RPL=    1 00002 00002
INFO  Date cols 11:15 packed 20970522  comp=(2097 05 22)
D - 970522    Õ†–¢
INFO  Date cols 11:14 packed 970522    comp=(1997 05 22)
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  12.31   PAGE    1
NEW:  D1       A        A                                    OLD:  D2       A        A

                      LINE COMPARE SUMMARY AND STATISTICS

      1 NUMBER OF LINE MATCHES              1  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
      0 REFORMATTED LINES                   1  PAIRED CHANGES (REFM+PAIRED INS/DEL)
      1 NEW FILE LINE INSERTIONS            0  NON-PAIRED INSERTS
      1 OLD FILE LINE DELETIONS             0  NON-PAIRED DELETES
      2 NEW FILE LINES PROCESSED
      2 OLD FILE LINES PROCESSED

LISTING-TYPE = DELTA  COMPARE-COLUMNS =    1:72     LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT)
THE FOLLOWING PROCESS STATEMENTS (USING COLUMNS 1:72) WERE PROCESSED:
   Y2PAST 1987
   NY2P 11:15 YYYYMMDD
   OY2P 11:14 YYMMDD
   NFOCUS COLS 1:20
   OFOCUS COLS 1:20
```

Figure 80. Example of Comparison Listing with Dates Being Compared

In Figure 80, the two date definition process statements have each caused an information (“INFO”) line to be generated. The information line shows:

- The position of the defined date in the record.
- The contents of the defined date field.
- The date as it was *actually compared*. In the second information line, you can see the defined date has a 2-digit year portion (“97”) but has actually been compared using a 4-digit year portion (“1997”).

For further details, see “Date Definitions” on page 263.

Note: Occasionally, you may see some “unusual” characters on the inserted (I) and deleted (D) lines. These characters simply represent data that is in a non-character (and therefore not directly printable) format in the input record and should be ignored.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  14.38   PAGE    1

NEW:  D1       A        A                                    OLD:  D2       A        A

                      LISTING OUTPUT SECTION (LINE COMPARE)

ID      SOURCE LINES                                                                  TYPE  LEN N-LN# O-LN#
    Account  Birth       Surname
    Number   Date                                                                     MAT=    1
I - 111222   19970101    Jones                                                       RPL=    1 00002 00002
D - 111222   970102      Jones
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  14.38   PAGE    1
NEW:  D1       A        A                                    OLD:  D2       A        A

                      LINE COMPARE SUMMARY AND STATISTICS

      1 NUMBER OF LINE MATCHES              1  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
      0 REFORMATTED LINES                   1  PAIRED CHANGES (REFM+PAIRED INS/DEL)
      1 NEW FILE LINE INSERTIONS            0  NON-PAIRED INSERTS
      1 OLD FILE LINE DELETIONS             0  NON-PAIRED DELETES
      2 NEW FILE LINES PROCESSED
      2 OLD FILE LINES PROCESSED

LISTING-TYPE = DELTA  COMPARE-COLUMNS =    1:72     LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT)
THE FOLLOWING PROCESS STATEMENTS (USING COLUMNS 1:72) WERE PROCESSED:
   COLHEAD 'Account','Number',1:8,N 1:6 C,O 1:6 C
   COLHEAD 'Birth','Date',10:20,N 7:11 P,O 7:10 P
   COLHEAD 'Surname',,22:61,N 12:51 C,O 11:50 C
```

Figure 81. Example of Comparison Listing with Column Headings (Using COLHEAD)

In Figure 81, COLHEAD process statements have been used to generate column headings (“Account Number,” “Birth Date,” and “Surname”) for the corresponding input data. For further details, see “Define Column Headings” on page 249.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  15.10
NEW:  JLEVERIN TEST2    A                                  OLD:  JLEVERIN TEST1

                      LISTING OUTPUT SECTION (LINE COMPARE)

ID          NEW FILE LINES          ID          OLD FILE LINES      N-LN# O-LN#
   ----+----1----+----2|...|5----+      ----+----1----+----2|...|5---
RN-This line is reforma|...|" file | RO-This line is reforma|...|"new 00001 00001
   This line is the sam|...|       |    This line is the sam|...|     00002 00002
I -This line differs fr|...|.      | D -This line differs fr|...|.    00003 00003
   This line is the sam|...|       |    This line is the sam|...|     00004 00004
I -This line is in the |...|ld".   |                        |...|     00005

1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  15.10
NEW:  JLEVERIN TEST2    A                                  OLD:  JLEVERIN TEST1

                  LINE COMPARE SUMMARY AND STATISTICS

   2 NUMBER OF LINE MATCHES            3  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
   1 REFORMATTED LINES                 2  PAIRED CHANGES (REFM+PAIRED INS/DEL)
   2 NEW FILE LINE INSERTIONS          1  NON-PAIRED INSERTS
   1 OLD FILE LINE DELETIONS           0  NON-PAIRED DELETES
   5 NEW FILE LINES PROCESSED
   4 OLD FILE LINES PROCESSED

LISTING-TYPE = CHNG   COMPARE-COLUMNS =    1:72     LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT) NARROW NOPRTCC

**ASMFSUPC INFORM04**, LISTING LINES MAY BE TRUNCATED DUE TO LIMITING OUTPUT LINE
WIDTH.
```

Figure 82. Example of a NARROW Side-by-Side Listing

In Figure 82, the *new* and *old* files are shown side-by-side. The NARROW listing type allows SuperC to output 55 columns from each file. Notice how the inserts and deletes are horizontally aligned with each other.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                                OLD:  JLEVERIN TEST1

                      LISTING OUTPUT SECTION (LINE COMPARE)

ID          NEW FILE LINES          ID          OLD FILE LINES      N-LN# O-LN#
   ----+----1----+----2|...|5----+      ----+----1----+----2|...|5---
RN-This line is reforma|...|" file | RO-This line is reforma|...|"new 00001 00001
   This line is the sam|...|        |                        |...|     00002 00002
I -This line differs fr|...|.       | D -This line differs fr|...|.    00003 00003
   This line is the sam|...|        |                        |...|     00004 00004
I -This line is in the |...|ld".    |                        |...|     00005
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                                OLD:  JLEVERIN TEST1

                      LINE COMPARE SUMMARY AND STATISTICS

   2 NUMBER OF LINE MATCHES             3  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
   1 REFORMATTED LINES                  2  PAIRED CHANGES (REFM+PAIRED INS/DEL)
   2 NEW FILE LINE INSERTIONS           1  NON-PAIRED INSERTS
   1 OLD FILE LINE DELETIONS            0  NON-PAIRED DELETES
   5 NEW FILE LINES PROCESSED
   4 OLD FILE LINES PROCESSED

LISTING-TYPE = CHNG   COMPARE-COLUMNS =    1:72     LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT) NARROW DLMDUP NOPRTCC

**ASMFSUPC INFORM04**, LISTING LINES MAY BE TRUNCATED DUE TO LIMITING OUTPUT LINE
WIDTH.
```

Figure 83. Example of a NARROW Side-by-Side Listing (with DLMDUP)

Figure 83, is similar to the previous example (Figure 82 on page 278) except that the process option DLMDUP has been used to suppress the matched lines from the *old* file section. This simplifies the combined listing output, allowing the changes to stand out more clearly.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                                OLD:  JLEVERIN TEST1

                      LISTING OUTPUT SECTION (LINE COMPARE)

ID          N|...|         ID          O|...|       TYPE LEN TYPE LEN N-LN# O-LN#
   ----+----1|...|+----8      ----+----1|...|+----8
RN-This line |...|000100 | RO-This line |...|000100 RFM=   1          00001 00001
   This line |...|000200 |              |...|       MAT=   1          00002 00002
I -This line |...|000300 | D -This line |...|000300 INS=   1 DEL=   1 00003 00003
   This line |...|000400 |              |...|       MAT=   1          00004 00004
I -This line |...|000500 |              |...|       INS=   1          00005
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                                OLD:  JLEVERIN TEST1

                      LINE COMPARE SUMMARY AND STATISTICS

   2 NUMBER OF LINE MATCHES           3  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
   1 REFORMATTED LINES                2  PAIRED CHANGES (REFM+PAIRED INS/DEL)
   2 NEW FILE LINE INSERTIONS         1  NON-PAIRED INSERTS
   1 OLD FILE LINE DELETIONS          0  NON-PAIRED DELETES
   5 NEW FILE LINES PROCESSED
   4 OLD FILE LINES PROCESSED

LISTING-TYPE = CHNG   COMPARE-COLUMNS =    1:72     LONGEST-LINE = 80
PROCESS OPTIONS USED: SEQ(DEFAULT) WIDE DLMDUP NOPRTCC
```

Figure 84. Example of a WIDE Side-by-Side Listing

In Figure 84, the *new* and *old* files are shown side-by-side in a WIDE listing. SuperC lists 80 columns from each file. Notice how the inserts and deletes are horizontally aligned with each other.

Note: The output file has a LRECL of 202/203 and may require special processing and printer capability to obtain a hard copy. Refer to the previous NARROW option examples if the large LRECL requirement cannot be satisfied and a side-by-side listing is still required.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (06/11/2004)  06/11/2004  16.46
 NEW:  PACKAGE   ANNOUNCE A1 + SELECTF FILE LIST               OLD:  PACKAGE

                      MEMBER SUMMARY LISTING (FILE COMPARE)

 DIFF SAME               FILE NAMES                  N-BYTES O-BYTES N-LINES

  **       PACKAGE   ANNOUNCE A1:PACKAGE ANNOUNCE E1     11210   51148     241
  **       PACKAGE   EXEC     A5:PACKAGE EXEC     E5    151749  151646    4311
       **  PACKAGE   HELP     A5:PACKAGE HELP     E5     70683   70683    1631
  **       PACKAGE   HELPCMS  A5:PACKAGE HELPCMS  E1        58      65       4
  **       PACKAGE   MENU     A5:PACKAGE MENU     E5     16803   16803     426
  **       PACKAGE   MODULE   A1:PACKAGE MODULE   E1    127604  126076       4
       **  PACKAGE   PACKAGE  A5:PACKAGE PACKAGE  E5      2408    2408      42
                     ----------------------          ------- ------- -------
                         GROUP TOTALS                 380515  418829    6659

Column 78 ------------------------------------->O-LINES N-HASH-SUM O-HASH-SUM

Column 80 ------------------------------------------>1147  C18E675F   F5CE6031
   (Continuation of previous data lines)            4310  2D2DF797   E0F1D820
                                                    1631  8A05CE27   8A05CE27
                                                       6  B1879676   F011E226
                                                     426  BAC0D5A9   BABFD5A9
                                                       4  4DF43D5A   3E820FA9
                                                      42  B29FA936   B29FA936
                                                 -------
                                                    7566


     7   TOTAL FILES PROCESSED AS A GROUP
     5   TOTAL FILES PROCESSED HAD CHANGES
     2   TOTAL FILES PROCESSED HAD NO CHANGES
     0   TOTAL NEW FILES NOT PAIRED
     0   TOTAL OLD FILES NOT PAIRED


 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SELECTF PACKAGE   ANNOUNCE A1   PACKAGE   ANNOUNCE E1
    SELECTF PACKAGE   EXEC     A5   PACKAGE   EXEC     E5
    SELECTF PACKAGE   HELP     A5   PACKAGE   HELP     E5
    SELECTF PACKAGE   HELPCMS  A5   PACKAGE   HELPCMS  E1
    SELECTF PACKAGE   MENU     A5   PACKAGE   MENU     E5
    SELECTF PACKAGE   MODULE   A1   PACKAGE   MODULE   E1
    SELECTF PACKAGE   PACKAGE  A5   PACKAGE   PACKAGE  E5
```

Figure 85. Example of a FILE Comparison of a File Group

Figure 85 shows a collection of files and statistics for the specified SELECTF designated file group. Some files are the same and some files differ.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
 NEW:  PACKAGE   ANNOUNCE A1 + SELECTF FILE LIST               OLD:  PACKAGE

                      MEMBER SUMMARY LISTING (FILE COMPARE)

 DIFF SAME               FILE NAMES                   N-BYTES O-BYTES N-LINES

  **       PACKAGE   ANNOUNCE A1:PACKAGE ANNOUNCE E1     11210   51148     241
  **       PACKAGE   EXEC     A5:PACKAGE EXEC     E5    151749  151646    4311
  **       PACKAGE   HELPCMS  A5:PACKAGE HELPCMS  E1        58      65       4
  **       PACKAGE   MENU     A5:PACKAGE MENU     E5     16803   16803     426
  **       PACKAGE   MODULE   A1:PACKAGE MODULE   E1    127604  126076       4
                     ----------------------          ------- ------- -------
                         GROUP TOTALS                 380515  418829    6659



Column 78 ------------------------------------->O-LINES N-HASH-SUM O-HASH-SUM

Column 80 ------------------------------------------>1147  C18E675F   F5CE6031
   (Continuation of previous data lines)            4310  2D2DF797   E0F1D820
                                                       6  B1879676   F011E226
                                                     426  BAC0D5A9   BABFD5A9
                                                       4  4DF43D5A   3E820FA9
                                                 -------
                                                    7566

     7   TOTAL FILES PROCESSED AS A GROUP
     5   TOTAL FILES PROCESSED HAD CHANGES
     2   TOTAL FILES PROCESSED HAD NO CHANGES
     0   TOTAL NEW FILES NOT PAIRED
     0   TOTAL OLD FILES NOT PAIRED

 PROCESS OPTIONS USED: LOCS

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SELECTF PACKAGE   ANNOUNCE A1   PACKAGE   ANNOUNCE E1
    SELECTF PACKAGE   EXEC     A5   PACKAGE   EXEC     E5
    SELECTF PACKAGE   HELP     A5   PACKAGE   HELP     E5
    SELECTF PACKAGE   HELPCMS  A5   PACKAGE   HELPCMS  E1
    SELECTF PACKAGE   MENU     A5   PACKAGE   MENU     E5
    SELECTF PACKAGE   MODULE   A1   PACKAGE   MODULE   E1
    SELECTF PACKAGE   PACKAGE  A5   PACKAGE   PACKAGE  E5
```

Figure 86. Example of a FILE Comparison of a File Group (with LOCS)

Figure 86 is similar to the previous example (Figure 85 on page 281) except that the LOCS process option has been used to limit the output to files from the file group which were found to be different. This option can greatly reduce the volume of output when the total number of files is secondary to the change activity in the group.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                   OLD:  JLEVERIN TEST1    A

                      LISTING OUTPUT SECTION (WORD COMPARE)

ID            SOURCE LINES (COMPARED COLUMNS)                       N-LN# O-LN#

   This line is reformatted; the spacing in the "new" file differs 00001 00001
   This line is the same in both files.                            00002 00002
MC-This line differs from the text in the       file.              00003 00003
IC-                                      "old"                    00003 00003
DC-                                      "new"                    00003 00003
   This line is the same in both files.                            00004 00004
I -This line is in the "new" file, but not in the "old".           00005 00004
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
NEW:  JLEVERIN TEST2    A                   OLD:  JLEVERIN TEST1    A

                  WORD COMPARE SUMMARY AND STATISTICS

 40 NUMBER OF WORD MATCHES      14  TOTAL CHANGES (PAIRED+NONPAIRED CHNG)
 14 NEW FILE WORD INSERTIONS     2  NEW FILE LINES CHANGED/INSERTED
  1 OLD FILE WORD DELETIONS      1  OLD FILE LINES CHANGED/DELETED
 54 NEW FILE WORDS PROCESSED     5  NEW FILE LINES PROCESSED
 41 OLD FILE WORDS PROCESSED     4  OLD FILE LINES PROCESSED

LISTING-TYPE = LONG   COMPARE-COLUMNS =    1:80     LONGEST-LINE = 80
PROCESS OPTIONS USED: NONE
```

Figure 87. Example of a WORD Comparison

Figure 87 is an output listing from a comparison using the WORD compare type and shows how the output lines differ when the comparison is made at the WORD level. The deleted words are normally listed under the replacement (inserted) words. Separate (both inserted and deleted) lines are listed when completely changed lines are detected.

The Search Listing

The typical search listing is composed of three parts:

- Page Heading
- Source Lines Section
- Summary Section

Page Heading

SuperC generates a page heading at the top of each page.

```
1 ASMFSUPC    -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0  (04/20/2004)  06/11/2004  12.32    PAGE     1
```

Figure 88. Example of the Page Heading Line for the Search Listing

Figure 88 shows a typical page heading line. It contains:

- Printer control page eject character (“1” in column one. not present when the NOPRTCC process option is specified).
- “Platform-identifier” One of “CMS,” “MVS,” or “VSE.”
- Program identification title including version and the version date: V1R5M0 (06/11/2004).

- The date and time of the search
- The page number.

Note: The program version and program date are important when reporting suspected SuperC problems.

Source Lines Section

The source lines section provides detailed information on the results of the Search.

```
1   LINE-#    SOURCE LINES                    SRCH FN:  NEW1  TESTCASE C

2         1     This NEW file is FIXED 80 with Sequence Numbers
          2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT
          3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT.
          4  ! * NEW: Use DPPDCMT for this comment.
          5  * * NEW: Use DPACMT to remove this assembler type comment
          6  -- *NEW: Use DPADCMT to remove this line.
          7    *   * NEW: COBOL comment.  Remove with DPCBCMT.
          8  C * NEW: FORTRAN comment.  Remove with DPFTCMT.
          9  &&& This NEW line comes out with a DPLINE '&&&'
```

Figure 89. Example of the Source Lines Section of a Search Listing

Figure 89 is an example showing the source line section. Only one character string (“NEW”) was specified for the search.

1 Column Header Line.

- **LINE-#** Relative line number of the line where the string was found.
- **SOURCE LINES** Up to 106 characters of the source line where the string was found.
- **SRCH FN:** Identifies the file which was searched. In this example, it is NEW1 TESTCASE C.

2 Text Lines. Relative line numbers and text lines from the search file where the string “NEW” was found.

The format of the source lines section changes when certain process options are used:

IDPFX (“Identifier Prefixed”**)**
The file ID is prefixed to each line of source text. See page 285.

LMTO (“List Group Member Totals”**)**
Only the totals of lines found and processed are listed. See page 285.

XREF (“Cross-reference Strings”**)**
Creates a cross-reference listing by search string. See page 286.

Note: the XREF process option also generates additional totals for each search string in the summary section.

Source Lines Section (IDPFX): The source line section generated when the IDPFX process option is used is similar to the normal source line section but with the search file ID preceding each line of source text. See Figure 90.

```
1  FNAME    FTYPE    FM  LINE-#  SOURCE-LNS   SRCH FN:  NEW1  TESTCASE C


2  NEW1     TESTCASE C1       1     This NEW file is FIXED 80 with Seque
   NEW1     TESTCASE C1       2  /** NEW: To get rid of this PLI/REXX type comm
   NEW1     TESTCASE C1       3  (** NEW: To get rid of this PASCAL type commen
   NEW1     TESTCASE C1       4  ! * NEW: Use DPPDCMT for this comment.
   NEW1     TESTCASE C1       5  * * NEW: Use DPACMT to remove this assembler t
   NEW1     TESTCASE C1       6  -- *NEW: Use DPADCMT to remove this line.
   NEW1     TESTCASE C1       7    *   * NEW: COBOL comment.  Remove with DPCBC
   NEW1     TESTCASE C1       8  C * NEW: FORTRAN comment.  Remove with DPFTCMT
   NEW1     TESTCASE C1       9  &&& This NEW line comes out with a DPLINE '&&&
```

Figure 90. Example of the IDPFX Source Lines Section of a Search Listing

1 Column Header Line.

- **FNAME** Filename (fn) of the file where in the string was found.
- **FTYPE** Filetype (ft) of the file where in the string was found.
- **FM** Filemode (fm) of the file where in the string was found.
- **LINE-#** Relative line number of the line where the string was found.
- **SOURCE-LNS** Up to 106 characters of the source line where the string was found.
- **SRCH FN:** In this example, the search file ID is NEW1 TESTCASE C.

2 The search file ID, relative line number, and text line from the search file where the string was found.

Source Lines Section (LMTO): The LMTO process option generates a listing showing the the total number of lines found and processed for each file. (The individual lines found are not listed.) See Figure 91.

```
1    FILES-SEARCHED          LINES-FOUND   LINES-PROC
2  NEW1     TESTCASE C1             9             9
   NEW13    TESTCASE C1            10            15
```

Figure 91. Example of the LMTO Source Lines Section of a Search Listing

1 Column Header Line.

- **FILES-SEARCHED** Identifies the files which were searched.
- **LINES-FOUND** Number of the lines found containing one or more of the search strings. The line is only counted once no matter how many search strings were found in the line.
- **LINES-PROC** Number of lines in the file that were searched. Does not include “Do not Process” lines.

2 Individual file totals.

Source Lines Section (XREF): The XREF process option creates a cross-reference listing where the source lines are listed by search strings.

In Figure 92, lines which contain the string “NEW” in NEW1 TESTCASE C1 are listed first, then lines which contain the string “NEW” in NEW13 TESTCASE C1, then lines which contain the string “USE” in NEW1 TESTCASE C1, and finally those lines which contain the string “USE” in NEW13 TESTCASE C1.

```
1 ----- STRING="NEW"              IN NEW1     TESTCASE C1 -----

2     1     This NEW file is FIXED 80 with Sequence Numbers
      2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
      3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)

⋮                                                                 . . .

3 -----                           IN NEW13    TESTCASE C1 -----

4     1     This NEW file is FIXED 80 with Sequence Numbers
      2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
      3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)

⋮                                                                 . . .

5 ----- STRING="USE"              IN NEW1     TESTCASE C1 -----

6     2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLC
      3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)
      4  ! * NEW: Use DPPDCMT for this comment.

⋮                                                                 . . .

7 -----                           IN NEW13    TESTCASE C1 -----

8     2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLC
      3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)
      4  ! * NEW: Use DPPDCMT for this comment.

⋮                                                                 . . .
```

Figure 92. Example of the XREF Source Lines Section (with ANYC)

1 Sub-section line showing string “NEW” and file NEW1 TESTCASE C1.
2 Line number and text of line where string was found.
3 Sub-section line showing file NEW13 TESTCASE C1 (string is still “NEW”).
4 Line number and text of line where string was found.
5 Sub-section line showing string “USE” and file NEW1 TESTCASE C1.
6 Line number and text of line where string was found.
7 Sub-section line showing file NEW13 TESTCASE C1 (string is still “USE”).
8 Line number and text of line where string was found.

Summary Section

The summary section (see Figure 93) provides various totals resulting from the search and shows any process statements which were used.

```
1   LINE-#    SOURCE LINES                    SRCH FN:  NEW1  TESTCASE C

2 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS
3         9            1             1             9         1: 80


4 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
  SRCHFOR   'NEW'
```

Figure 93. Example of the Summary Section of a Search Listing

The summary section consists of:

1 A section heading line.

2 A column heading line.

3 One line of totals.

4 A multi-line section (two lines in Figure 93) showing the process statements which were used.

XREF Summary Section: When the XREF process option ("Cross-reference Strings") is used, additional lines are included in the summary section. In Figure 94, these are lines **2** , **3** , and **4** . The totals are listed according to each search string.

Note: The XREF summary section may be produced without the XREF source line section by using the LMTO process option.

```
1          SUMMARY SECTION                  SRCH FN:  NEW1*  TESTCASE C

2 STRING-FOUND           LINES-FOUND   FILES-W/LNS    STRING-NOT-FOUND
3      "NEW"                19             2
4      "USE"                10             2

5 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS
6        29            2             5            64         1: 80
```

Figure 94. Example of the XREF Summary Section of a Search Listing

1 Section header line. Identifies the file which was searched. In this example, it is NEW1* TESTCASE C.

2 Column header line.

STRING-FOUND
Column indicating the search string.

LINES-FOUND Lines which contained one or more occurrences of the search string.

FILES-W/LNS Total number of files in the group in which the string was found.

STRING-NOT-FOUND
Indication that the string was not found in any of the files in the file group.

3 Totals for string “NEW”

4 Totals for string “USE”

5 Column header line.

LINES-FOUND The summation of lines found for the individual search strings.

FILE-W/LNS Number of files where lines were found to contain one or more of the search strings.

FILE-PROC Number of files that were searched.

LINES-PROC Number of lines that were part of the search set.

COMPARE-COLS The column range that was searched.

6 Totals statistics arranged under the columns specified in **5**.

Examples of Search Listings

Search of One File: Figure 95 shows the 3 parts of a search listing: page heading, source lines section, and summary section.

```
1       ASMFSUPC - CMS LINE/WORD/BYTE COMPARE UTILITY - V1R5M0  (06/11/2004)
   LINE-#    SOURCE LINES                   SRCH FN:  NEW1     TESTCASE C

       2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
       3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)
       5  * * NEW: Use DPACMT to remove this assembler type comment.
       6  -- *NEW: Use DPADCMT to remove this line.

1       ASMFSUPC - CMS LINE/WORD/BYTE COMPARE UTILITY - V1R5M0  (06/11/2004)  06/11/2004
          SUMMARY SECTION                   SRCH FN:  NEW1     TESTCASE C

 LINES-FOUND   FILES-W/LNS   FILES-PROC  LINES-PROC  COMPARE-COLS  LONGEST-LINE
         4           1            1            9         1: 80           80


 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'
```

Figure 95. Example of the Search Listing (Single File)

IDPFX Search of File Group: The file group NEW1* TESTCASE C composed of 5 files was searched and 8 lines within 2 files had “remove” and “rid” as the search arguments.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
 FNAME    FTYPE    FM  LINE-#  SOURCE-LNS   SRCH FN:  NEW1*    TESTCASE C


 NEW1     TESTCASE C1       2  /** NEW: To get rid of this PLI/REXX type comment
 NEW1     TESTCASE C1       3  (** NEW: To get rid of this PASCAL type comment,
 NEW1     TESTCASE C1       5  * * NEW: Use DPACMT to remove this assembler type
 NEW1     TESTCASE C1       6  -- *NEW: Use DPADCMT to remove this line.

 NEW13    TESTCASE C1       2  /** NEW: To get rid of this PLI/REXX type comment
 NEW13    TESTCASE C1       3  (** NEW: To get rid of this PASCAL type comment,
 NEW13    TESTCASE C1       5  * * NEW: Use DPACMT to remove this assembler type
 NEW13    TESTCASE C1       6  -- *NEW: Use DPADCMT to remove this line.

1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
          SUMMARY SECTION                   SRCH FN:  NEW1*    TESTCASE C

 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS  LONGEST-LIN
         8            2             5            64         1: 80            80

 PROCESS OPTIONS USED: IDPFX

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'
```

Figure 96. Example of IDPFX Search on File Group

XREF Search of File Group for Two Strings: XREF sorts the search string occurrences before producing a listing. In the example shown below both strings were found in the file group.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
   LINE-#    SOURCE LINES                   SRCH FN:  NEW1*    TESTCASE C


 ----- STRING="remove"           IN NEW1     TESTCASE C1 -----

       5  * * NEW: Use DPACMT to remove this assembler type comment.
       6  -- *NEW: Use DPADCMT to remove this line.

 -----                            IN NEW13    TESTCASE C1 -----

       5  * * NEW: Use DPACMT to remove this assembler type comment.
       6  -- *NEW: Use DPADCMT to remove this line.

 ----- STRING="rid"              IN NEW1     TESTCASE C1 -----

       2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
       3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)

 -----                            IN NEW13    TESTCASE C1 -----

⋮

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'
```

Figure 97. Example of XREF Search on File Group for Two Strings

LMTO Search of File Group: LMTO produces only the summary section for the search operation.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
     FILE TOTALS SECTION                   SRCH FN:  NEW1*    TESTCASE C

   FILES-SEARCHED        LINES-FOUND   LINES-PROC
 NEW1     TESTCASE C1            4            9
 NEW13    TESTCASE C1            4           15

1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
          SUMMARY SECTION                   SRCH FN:  NEW1*    TESTCASE C

 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS  LONGEST-LIN
         8            2             5            64         1: 80            80

 PROCESS OPTIONS USED: LMTO

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'
```

Figure 98. Example of LMTO Search on File Group

LMTO Search of File Group Using the XREF Process Option: This is another example of a summary only output. Contrasting with the previous example, the string totals are sorted before being listed.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
     XREF TOTALS SECTION                   SRCH FN:  NEW1*    TESTCASE C

     STRING-USED          FILES-SEARCHED         LINES-FOUND    LINES-PROC
     "remove"             NEW1     TESTCASE C1       2            10
                          NEW13    TESTCASE C1       2            16

     "rid"                NEW1     TESTCASE C1       2            10
                          NEW13    TESTCASE C1       2            16

1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
          SUMMARY SECTION                  SRCH FN:  NEW1*    TESTCASE C

 STRING-FOUND           LINES-FOUND   FILES-W/LNS       STRING-NOT-FOUND
      "remove"              4             2
      "rid"                 4             2

 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS  LONGEST-LIN
         8            2             5            64         1: 80            80

 PROCESS OPTIONS USED: LMTO XREF

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'

 .
```

Figure 99. Example of XREF/LMTO Search of File Group

LTO Search of File Group: LTO produces the overall totals section of the search results.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
          SUMMARY SECTION                  SRCH FN:  NEW1*    TESTCASE C

 LINES-FOUND    FILES-W/LNS    FILES-PROC  LINES-PROC  COMPARE-COLS  LONGEST-LIN
         8            2             5            64         1: 80            80

 PROCESS OPTIONS USED: LTO

 THE FOLLOWING PROCESS STATEMENT(S) WERE PROCESSED:
    SRCHFOR   'remove'
    SRCHFOR   'rid'
```

Figure 100. Example of LTO Search on File Group

LPSF Search of File Group: The process option LPSF ("List Previous-Search-Following Lines") lists lines before and after the search text detected line. The "*" in the line number column indicate they were part of the extra lines listed.

```
1  ASMFSUPC   -     CMS FILE/LINE/WORD/BYTE/SFOR COMPARE UTILITY- V1R5M0
   LINE-#    SOURCE LINES                   SRCH FN:  NEW1*    TESTCASE C


 NEW1     TESTCASE C1 -------------- STRING(S) FOUND --------

       *     This line is reformatted; the spacing in the "new" member differs.
       2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
       3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)
       *  ! * NEW: Use DPPDCMT for this comment.
       5  * * NEW: Use DPACMT to remove this assembler type comment.
       6  -- *NEW: Use DPADCMT to remove this line.
       *    *   * NEW: COBOL comment.  Remove with DPCBCMT.
       *  C * NEW: FORTRAN comment.  Remove with DPFTCMT.
       *  &&& This NEW line comes out with a DPLINE '&&&'

 NEW13    TESTCASE C1 -------------- STRING(S) FOUND --------

       *     This NEW file is FIXED 80 with Sequence Numbers
       2  /** NEW: To get rid of this PLI/REXX type comment, use DPPLCMT. */
       3  (** NEW: To get rid of this PASCAL type comment, use DPPSCMT. *)
```

Figure 101. Example of LPSF Search on File Group

Update Files

An update file contains information relating to the result of a comparison and is generated when one of the *update* process options is specified:

UPDCMS8 ("Update CMS Sequenced 8 File" on page 295)
UPDCNTL ("Update Control Files" on page 296)
UPDLDEL ("Update Long Control" on page 299)
UPDMVS8 ("Update MVS Sequenced 8 File" on page 300)
UPDPDEL ("Update Prefixed Delta Lines" on page 301)
UPDREV ("Revision File" on page 293)
UPDREV2 ("Revision File (2)" on page 294)
UPDSEQ0 ("Update Sequenced 0 File" on page 301)
UPDSUMO ("Update Summary Only Files" on page 302)

Notes:

1. UPDCMS8, UPDMVS8, UPDPDEL, UPDREV, UPDREV2, and UPDSEQ0 do not generate an update file after a comparison of matching files (Return Code = 0).
2. For details on the *naming* of the update file:
   - On MVS, see "Invoking the Comparison on MVS" on page 180.
   - On CMS, see "Update File ID" on page 190.
   - On VSE, see "Invoking the Comparison on VSE" on page 199.

- Dates, where applicable, in the heading lines of update files are in the format MM/DD/YYYY.

3. All “do not process” options, and DPLINE or CMPLINE process statements are invalid when used with the process options UPDCMS8, UPDMVS8, UPDSEQ0, UPDLDEL and UPDPDEL. The “do not process” options are cancelled with error notification ASMF014.

Update files are normally used as input to post-processing programs and can be specified besides the normal listing output file.

On the following pages, descriptions and examples are given of the contents of the update file produced for each of the update (UPD...) process options.

In most of the examples shown, the same two input files are used. The contents of the *old* file are shown in Figure 102. The contents of the *new* file are shown in Figure 103.

```
This line is reformatted; the  spacing  in  the  "new"  file  differs.  00000100
This line is the same in both files.                                     00000200
This line differs from the text in the "new" file.                       00000300
This line is the same in both files.                                     00000400
```

Figure 102. The “Old” Input File Used in Most of the Update Examples

```
This line is reformatted; the spacing in the "new" file differs.         00000100
This line is the same in both files.                                     00000200
This line differs from the text in the "old" file.                       00000300
This line is the same in both files.                                     00000400
This line is in the "new" file, but not in the "old".                    00000500
```

Figure 103. The “New” Input File Used in Most of the Update Examples

Revision File

The process option UPDREV produces an update file containing a copy of the *new* source text with revision tags delimiting the changed text lines.

The UPDREV process option is available for LINE and WORD compare types.

UPDREV supports two different types of revision tags, one for SCRIPT/VS files and one for IBM BookMaster files. (Use the REVREF process statement (“Revision Code Reference” on page 255) to specify which type of revision tag you want.)

Figure 104 on page 294 shows a SuperC UPDREV file with SCRIPT/VS revision tags (.rc on/off).

```
.rc 1 |
.rc 1 on
This line is reformatted; the spacing in the "new" file differs.
.rc 1 off
This line is the same in both files.
.rc 1 on
This line differs from the text in the "old" file.
.rc 1 off
This line is the same in both files.
.rc 1 on
This line is in the "new" file, but not in the "old".
.rc 1 off
```

Figure 104. Example of a UPDREV Update File for SCRIPT/VS Documents

When the UPDREV update file in Figure 104 is subsequently processed by SCRIPT/VS, the final scripted output has "|" revision characters in the left margin of the output document identifying the changed lines (those between the SCRIPT/VS revision tags `.rc 1 on` and `.rc 1 off`).

Note: The revision character ("|" in the example in Figure 104) can be specified either by using a REVREF process statement (see "Revision Code Reference" on page 255) or by having a SCRIPT/VS `.rc` revision tag as the first record in the *new* file. Subsequent changes to the source can therefore be separately identified by using different revision characters.

Figure 105 shows a SuperC UPDREV file with IBM BookMaster revision tags (:rev/:erev).

```
This line is reformatted; the spacing in the "new" file differs.
This line is the same in both files.
This line differs from the text in the "old" file.
This line is the same in both files.
This line is in the "new" file, but not in the "old".
```

Figure 105. Example of a UPDREV Update File for IBM BookMaster Documents

When the UPDREV update file in Figure 105 is subsequently processed by IBM BookMaster, the final formatted output has the revision character associated with the revision ID `abc` (as specified by a `:revision.` IBM BookMaster tag in the *new* input file) in the left margin of the output document identifying the changed lines (those between the IBM BookMaster revision tags `:rev` and `:erev`).

Note: The revision ID (`abc` in the example in Figure 105) is controlled by the REVREF process statement (see "Revision Code Reference" on page 255). Subsequent changes to the source can therefore be separately identified by using different revision IDs (which are associated with unique revision characters).

Revision File (2)

The process option UPDREV2 is identical to UPDREV with the exception that data between the following IBM BookMaster tags are not deleted in the update file:

```
:cgraphic.
:ecgraphic.

:fig.
:efig.
```

```
:lblbox.
:elblbox.

:nt.
:ent.

:screen.
:escreen.

:table.
:etable.

:xmp.
:exmp.
```

Update CMS Sequenced 8 File

The process option UPDCMS8 produces update files that are generally created for input to the CMS UPDATE command. The CMS UPDATE command is described in the *CMS Command Reference* manual.

The UPDCMS8 process option is available for the LINE compare type only.

The *old* input file must have fixed-length 80-byte records with valid sequence numbers in columns 73 through 80. The *new* file must be fixed but may have a length less than or equal to 80.

The UPDCMS8 update file is fixed-length 80.

If the sequence numbers do not allow adequate room to insert changes from the *new* file, SuperC changes the status of adjacent matched lines to find the room.

UPDCMS8 update files contain both CMS UPDATE control statements and source lines from the “new” file. All UPDCMS8 control statements are identified by the characters “./” in columns 1 and 2 of the 80-byte record, followed by one or more spaces and a one-character control line identifier. The control line identifiers are sequence (S), insert (I), delete (D), replace (R), and comment (*). Figure 106 shows an example of a UPDCMS8 update file.

```
1 ./ * NEW:  JLEVERIN TEST2    A                          06/11/2004 11.35
2 ./ * OLD:  JLEVERIN TEST1    A
3 ./ R 00000100 00000100 $ 00000140 00000040
4 This line is reformatted; the spacing in the "new" file differs.     00000100
5 ./ R 00000300 00000300 $ 00000340 00000040
6 This line differs from the text in the "old" file.                   00000300
7 ./ I 00000400          $ 00001400 00001000
8 This line is in the "new" file, but not in the "old".                00000500
```

Figure 106. Example of a UPDCMS8 Update File

The example in Figure 106, has the following lines:

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Replacement control line. The first 8-digit numeric field is the sequence number (of the *old* file) of the first input number to be replaced. The second 8-digit numeric field is the sequence number of the *old* file that is the last record to be replaced. The dollar sign is an option separator field. The third and fourth 8-digit fields represent the first decimal number to be used for sequencing the substitute records and the decimal increment to be used in the sequencing, respectively.

In this example, the first line of the *old* file is being replaced with one line from the *new* file.

4 The *new* record which has replaced the *old* record at sequence number 00000100.

5 Another replacement control line.

6 The *new* record which has replaced the *old* record at sequence number 00000300.

7 Insert control line. After *old* line 4, there is a line inserted in the *new* file.

8 The text of the inserted line.

Update Control Files

The process option UPDCNTL produces a control file that relates matches, insertions, deletions, and reformats to:

- The relative line numbers of the *old* and *new* files (LINE compare type); see Figure 107.
- The relative word position of the *old* file (WORD compare type); see Figure 108 on page 297.
- The relative byte offset (BYTE compare type); see Figure 109 on page 298.

Note: No source or data from either input file is included in the update file produced by UPDCNTL.

Update Control File (LINE Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                                  06/11/2004 12.45
2 *    OLD:  JLEVERIN TEST1    A
3 *  N-LINE-# O-LINE-# MAT-LEN  INS-LEN  DEL-LEN REFM-LEN
4    00000001 00000001                          00000001
5    00000002 00000002 00000001
6    00000003 00000003          00000001 00000001
7    00000004 00000004 00000001
8    00000005 00000005          00000001
9 *  END
```

Figure 107. Example of a UPDCNTL Update File Using LINE Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Header Comment line. For information on the columns, see Table 27 on page 297.

4 Shows that line 1 of the *new* file is a reformatted line of line 1 of the *old* file.

5 Line 2 from both files match.

6 Line 3 of the *new* file replaces line 3 of the *old* file.

7 Line 4 from both lines match.

8 At line 5 of the *new* file is an inserted line.

9 Comment line. This is the end of the update file.

The following table shows the column numbers used for the UPDCNTL file:

Table 27. UPDCNTL Update File Format Using LINE Compare Type

Column #	Identifier	Data Item
4-11	N-LINE-#	*New* line number
13-20	O-LINE-#	*Old* line number
22-29	MAT-LEN	Match length
31-38	INS-LEN	Insert length
40-47	DEL-LEN	Delete length
49-56	REFM-LEN	Reformat length
58-65	N-DP-LEN	(Not shown) *New* “Do not Process” length
67-74	O-DP-LEN	(Not shown) *Old* “Do not Process” length
76-80	N-MVL	(Not shown) *New* “moved” line length.

Update Control File (WORD Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                                06/11/2004 12.17
2 *    OLD:  JLEVERIN TEST1    A
3 *  N-LINE-# N-LN-LEN N-COL WD-MAT-# N-WD-INS O-WD-DEL O-LINE-# O-LN-LEN O-COL
4    00000001 00000003 00001 00000027                   00000001 00000003 00001
5    00000003 00000001 00040          00000001 00000001 00000003 00000001 00040
6    00000003 00000002 00046 00000009                   00000003 00000002 00046
7    00000005 00000001 00001          00000013
8 *  END
```

Figure 108. Example of a UPDCNTL Update File Using WORD Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Header comment line. For information on the columns, see Table 28 on page 298.

4 Beginning with line one column 1, of both files, the first twenty-seven words match. This takes us to line 3.

5 There is 1 word replaced in line 3. It begins in column forty of each file.

6 Beginning from the change in **5**, there are 9 more words that match.

7 A line of thirteen words was inserted at line 5.

8 Comment line. Ends the update file.

The following table shows the column numbers used for the UPDCNTL file:

Table 28. UPDCNTL Update File Format Using WORD Compare Type

Column #	Identifier	Data Item
4-11	N-LINE-#	Beginning *new* line number
13-20	N-LN-LEN	Number of lines
22-26	N-COL	*New* column number (beginning of word)
28-35	WD-MAT-#	Number of matching words
37-44	N-WD-INS	Number of *new* inserted words
46-53	O-WD-DEL	Number of *old* deleted words
55-62	O-LINE-#	Beginning *old* line number
64-71	O-LN-LEN	Number of *old* lines
73-77	O-COL	*Old* column number

Update Control File (BYTE Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                              06/11/2004 12.23
2 *    OLD:  JLEVERIN TEST1    A
3 *  N-BYTE-O O-BYTE-O MAT-LEN  INS-LEN  DEL-LEN
4    00000000 00000000 0000001E
5    0000001E 0000001E                   00000001
     0000001E 0000001F 00000008
     00000026 00000027                   00000001
     00000026 00000028 00000002
     00000028 0000002A                   00000001
     00000028 0000002B 00000004
     0000002C 0000002F                   00000001
     0000002C 00000030 00000007
     00000033 00000037                   00000001
     00000033 00000038 00000004
     00000037 0000003C                   00000001
     00000037 0000003D 00000009
     00000040 00000046          00000006
     00000046 00000046 00000082
6    000000C8 000000C8          00000003 00000003
     000000CB 000000CB 00000075
7    00000140 00000140          00000050
8  * END
```

Figure 109. Example of a UPDCNTL Update File Using BYTE Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Header comment line. For more information on the columns, see Table 29 on page 299.

4 First thirty-one (1E hex) bytes match.

5 1 byte is deleted.

6 (Skipping several lines). 3 bytes of the *new* file replace 3 bytes of the *old* file.

7 Fifty bytes inserted.

8 Comment line. Ends the update file.

The following table shows the column numbers used for the UPDCNTL file:

Table 29. UPDCNTL Update File Format Using BYTE Compare Type

Column #	Identifier	Data Item
4-11	N-BYTE-O	*New* byte offset
13-20	O-BYTE-O	*Old* byte offset
22-29	MAT-LEN	Number of matching bytes
31-38	INS-LEN	Number of inserted bytes
40-47	DEL-LEN	Number of deleted bytes

Update Long Control

The process option UPDLDEL produces an update file that contains control records, matching *new* file source records, inserted *new* file source records, and deleted *old* file source records.

The UPDLDEL process option is available for the LINE compare type only.

Figure 110 on page 300 shows an example of a UPDLDEL update file.

The control records are titled as follows:

***HDR1, *HDR2, *HDR3**
Header titles and data

***M-** Matched line sequence header

***I-** Inserted line sequence header

***I-RP** Inserted line sequence header for replacement lines

***I-RF** Inserted line sequence header for reformatted lines

***D-** Deleted line sequence header

***D-RP** Deleted line sequence header for replacement lines

***D-RF** Deleted line sequence header for reformatted lines

Header control records are full length records that delimit the copied file records. This allows you to quickly find changed areas. The records appear similar to the information on a LONG listing. The two input files must both have the same fixed record length or each have a variable record length.

```
*HDR1  JLEVERIN   TEST2    A                                  06/11/2004 14.58
*HDR2  JLEVERIN   TEST1    A                                  TYPE = UPDLDEL
*I-RF  INS#=  1      N-REF#=000001 O-REF#=000001  *****ASMFSUPC CHANGE HEADER*****
This line is reformatted; the spacing in the "new" file differs.        00000100
*D-RF  DEL#=  1      N-REF#=000001 O-REF#=000001  *****ASMFSUPC CHANGE HEADER*****
This line is reformatted; the  spacing  in  the  "new"  file  differs.  00000100
*M-    MAT#=  1      N-REF#=000002 O-REF#=000002  *****ASMFSUPC CHANGE HEADER*****
This line is the same in both files.                                    00000200
*I-RP  INS#=  1      N-REF#=000003 O-REF#=000003  *****ASMFSUPC CHANGE HEADER*****
This line differs from the text in the "old" file.                      00000300
*D-RP  DEL#=  1      N-REF#=000003 O-REF#=000003  *****ASMFSUPC CHANGE HEADER*****
This line differs from the text in the "new" file.                      00000300
*M-    MAT#=  1      N-REF#=000004 O-REF#=000004  *****ASMFSUPC CHANGE HEADER*****
This line is the same in both files.                                    00000400
*I-    INS#=  1      N-REF#=000005 O-REF#=000004  *****ASMFSUPC CHANGE HEADER*****
This line is in the "new" file, but not in the "old".                   00000500
```

Figure 110. Example of a UPDLDEL Update File

Update MVS Sequenced 8 File

The process option UPDMVS8 produces a file that contains both control records and *new* file source lines using sequence numbers from *old* file columns 73 to 80.

The UPDMVS8 process option is available for the LINE Compare Type only.

The format of the generated data may be suitable as input to the IEBUPDTE utility. See *OS/VS2 MVS Utilities* for information about the contents of this file. Figure 111 shows an example of a UPDMVS8 update file created on CMS.

```
1 ./  CHANGE LIST=ALL OLD:JLEVERIN  TEST1   A
2 ./ DELETE  SEQ1=00000100,SEQ2=00000100
3 This line is reformatted; the spacing in the "new" file differs.        00000100
4 ./ DELETE  SEQ1=00000300,SEQ2=00000300
5 This line differs from the text in the "old" file.                      00000300
6 This line is in the "new" file, but not in the "old".                   00000500
```

Figure 111. Example of a UPDMVS8 Update File

1 Control record. Lists *old* file name.

2 Control record. Shows record deleted at sequence number 100 on the *old* file.

3 Inserted line from the *new* file.

4 Control record. Shows record deleted at sequence number 300 on the *old* file.

5 Inserted line from the *new* file.

6 Inserted line from the *new* file.

The files to be compared must have fixed-length 80-byte records. They must also contain sequence numbers.

Update Prefixed Delta Lines

The process option UPDPDEL produces a variable-length update file that contains header records and complete delta lines from the input files, up to a maximum of 32K bytes in each output line.

The UPDPDEL process option is available for the LINE compare type only.

Figure 112 shows an example of a UPDPDEL update file.

Prefix codes (`I` for insert and `D` for delete) together with the line number precede lines from the input files. Sub-totals are shown before each group of flagged records:

- `INS#=` for the number of consecutive inserted records,
- `DEL#=` for the number of consecutive deleted records,
- `RPL#=` for the number of consecutive *pairs* of replaced records, and
- `MAT#=` for the number of intervening matched records.

```
1 *    NEW:   JLEVERIN   TEST2    A                              06/11/2004 12.08
2 *    OLD:   JLEVERIN   TEST1    A
3 *ID-   LINE#       SOURCE LINE
4 *                                  RPL#= 00000001
5 I - 00000001  This line is reformatted; the spacing in the "new" file differs.        00000100
6 D - 00000001  This line is reformatted; the  spacing  in  the  "new"  file  differs.  00000100
4 *                                  RPL#= 00000001               MAT#= 00000001
5 I - 00000003  This line differs from the text in the "old" file                       00000300
6 D - 00000003  This line differs from the text in the "new" file.                      00000300
7 *                                  INS#= 00000001               MAT#= 00000001
8 I - 00000005  This line is in the "new" file, but not in the old.
9 *   END                                                                               00000500
```

Figure 112. Example of a UPDPDEL Update File

The example in Figure 112 has the following lines:

1. Comment line. Lists the *new* file name and the date and time of the comparison.
2. Comment line. Lists the *old* file name.
3. Header comment line.
4. Sub-total line showing that 1 replaced *pair* of records follow.
5. The line that has replaced the line in the *old* file.
6. The line in the *old* file that has been replaced.
7. Sub-total line showing that 1 inserted record follows.
8. The line that has been inserted in the *new* file.
9. Comment line. Ends the update file.

Update Sequenced 0 File

The process option UPDSEQ0 produces a control file that relates insertions and deletions to the relative line numbers of the *old* file. UPDSEQ0 is similar to UPDCMS8, but uses relative line numbers instead of sequence numbers from the *old* file.

The UPDSEQ0 process option is available for the LINE compare type only.

This update file is characterized by control statements followed by source lines from the *new* file. All UPDSEQ0 control statements are identified by the characters "**./**" in columns 1 and 2 of the 80-byte record, followed by one or more spaces and additional space-delimited fields. The control statements are insert (I), delete (D), replace (R), and comment (*). Control statement data does not extend beyond column 50. Figure 113 shows an example of a UPDSEQ0 update file.

```
1 ./ * NEW:  JLEVERIN TEST2    A                          06/11/2004 13.34
2 ./ * OLD:  JLEVERIN TEST1    A
3 ./ R 00000001 00000001 $ 00000001
4 This line is reformatted; the spacing in the "new" file differs.     00000100
5 ./ R 00000003 00000003 $ 00000001
6 This line differs from the text in the "old" file.                   00000300
7 ./ I 00000004          $ 00000001
8 This line is in the "new" file, but not in the "old".                00000500
```

Figure 113. Example of a UPDSEQ0 Update File

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Replacement control record. Beginning at the first record of the *old* file, replace 1 record. The numeric value after the dollar sign specifies the number of *new* file source lines that follow the control record.

4 Text of *new* file line to replace line 1.

5 Replace the third record with 1 record.

6 Text of *new* file line to replace line 3.

7 Insert control line. Insert 1 line after record 4 of *old* file.

8 Text of inserted line.

Update Summary Only Files

The process option UPDSUMO produces an update file of 4 lines: *new* file name, *old* file name, column headers, and a summary totals line.

The UPDSUMO process option is available for the LINE, WORD, and BYTE compare types.

The summary totals line has a "T" in column 1. The summary statistics are located at fixed offsets in the output line. The file has a line length of 132 bytes.

Update Summary Only File (LINE Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                                   06/11/2004
2 *    OLD:  JLEVERIN TEST1    A
3 *  NEW-PROC OLD-PROC NEW-INS  OLD-DEL  TOT-CHG  TOT-RFM  FI-PROC  FI-DIFF
4 T  00000005 00000004 00000002 00000001 00000003 00000001 00000001 00000001

 . .  (Continuation of previous data lines)  . . . . . .

1 13.39
2
3 N-NOT-PD O-NOT-PD N-DP-LNS O-DP-LNS
4 00000000 00000000 00000000 00000000
```

Figure 114. Example of a UPDSUMO File Using LINE Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Comment line. Header line. Columns are explained in Table 30.

4 Totals line.

In Figure 114, the update summary file is shown in split screen mode. The bottom half of the screen shows the result of scrolling right to see the remainder of the member.

The following table shows the column numbers used to display the update information:

Table 30. UPDSUMO Format Using LINE Compare Type

Column #	Identifier	Data Item
	NEW-PROC	Number of *new* lines processed
	OLD-PROC	Number of *old* lines processed
	NEW-INS	Number of *new* line insertions
	OLD-DEL	Number of *old* line deletions
	TOT-CHG	Total number of line changes
	TOT-RFM	Total number of reformats
	FI-PROC	Total number of files/members processed
	FI-DIFF	Total number of files/members different
	N-NOT-PD	Total *new* files/members not processed
	O-NOT-PD	Total *old* files/members not processed
	N-DP-LNS	Total number of *new* “do not process” lines
	O-DP-LNS	Total number of *old* “do not process” lines

Update Summary Only File (WORD Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                                    06/11/2004
2 *    OLD:  JLEVERIN TEST1    A
3 *  NEW-PROC OLD-PROC NEW-INS  OLD-DEL  TOT-CHG           FI-PROC  FI-DIFF
4 T  00000049 00000037 00000013 00000001 00000013          00000001 00000001

 . .  (Continuation of previous data lines)  . . . . . . . . . . . . . .

1       13.48
2
3       N-NOT-PD O-NOT-PD
4       00000000 00000000
```

Figure 115. Example of a UPDSUMO File Using WORD Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Comment line. Header line. Columns are explained in Table 31.

4 Totals line.

In Figure 115, the UPDSUMO file is shown in split screen mode. The bottom half of the screen is scrolled right to show the remainder of the member.

The following table shows the column numbers used to display the update information:

Table 31. UPDSUMO Format Using WORD Compare Type

Column #	Identifier	Data Item
	NEW-PROC	Number of *new* words processed
	OLD-PROC	Number of *old* words processed
	NEW-INS	Number of *new* word insertions
	OLD-DEL	Number of *old* word deletions
	TOT-CHG	Total number of word changes
	FI-PROC	Total number of files/members processed
	FI-DIFF	Total number of files/members different
	N-NOT-PD	Total *new* files/members not processed
	O-NOT-PD	Total *old* files/members not processed

Update Summary Only File (BYTE Compare Type)

```
1 *    NEW:  JLEVERIN TEST2    A                                  06/11/2004
2 *    OLD:  JLEVERIN TEST1    A
3 *  NEW-PROC OLD-PROC NEW-INS  OLD-DEL  TOT-CHG          FI-PROC  FI-DIFF
4 T  00000400 00000320 00000089 00000009 00000095         00000001 00000001

 . .  (Continuation of previous data lines)  . . . . . .  . . . . . .

 COMMAND ===> _                                            SCROLL ===> PAGE
1      13.51
2
3      N-NOT-PD O-NOT-PD
4      00000000 00000000
```

Figure 116. Example of a UPDSUMO File Using BYTE Compare Type

1 Comment line. Lists the *new* file name and the date and time of the comparison.

2 Comment line. Lists the *old* file name.

3 Comment line. Header line. Columns are explained in Table 32.

4 Totals line.

In Figure 116, the UPDSUMO file is shown in split screen mode. The bottom half of the screen shows the result of scrolling right to see the remainder of the member.

The following table shows the column numbers used to display the update information:

Table 32. UPDSUMO Format Using BYTE Compare Type

Column #	Identifier	Data Item
	NEW-PROC	Number of *new* bytes processed
	OLD-PROC	Number of *old* bytes processed
	NEW-INS	Number of *new* byte insertions
	OLD-DEL	Number of *old* byte deletions
	TOT-CHG	Total number of byte changes
	FI-PROC	Total number of files/members processed
	FI-DIFF	Total number of files/members different
	N-NOT-PD	Total *new* files/members not processed
	O-NOT-PD	Total *old* files/members not processed

CMS File Selection List

When you are dealing with a *group* of CMS files (or library members), some or all of which you want to use as input to a comparison or search, you need to use the CMS file selection list.

In the case of a comparison, the CMS file selection list enables you to specify which files (or members) in the *new* group are to be compared with which files (or members) in the *old* group.

In the case of a search, the CMS file selection list enables you to specify which files (or members) in the group are to be searched.

To view the CMS file selection list for the files (or members) that you want to compare or search, you use the Selection List Menu.

Getting to the Selection List Menus

There are two formats of the Selection List Menu. One format is for comparisons and the other format is for searches. (Each format varies slightly depending on whether you are dealing with a group of files or a group of library members.)

The appropriate format of the Selection List Menu is displayed after you have entered the relevant *group* details in either the Primary Comparison Menu or the Primary Search Menu:

- The Selection List field must contain an "*" (asterisk)
- For a comparison of a group of files, the New File ID or the Old File ID must contain an "*"
- For a comparison of a MACLIB or TXTLIB, the Member Name must contain an "*"
- For a search of a group of files, the Search File ID must contain an "*"
- For a search of a MACLIB or TXTLIB, the Member Name must contain an "*"

After you have entered the above details in the Primary Comparison Menu or the Primary Search Menu, press Enter to display the corresponding Selection List Menu.

In the case of a comparison, SuperC creates two lists: one list of the files (or members) in the *new group* and one list of the files (or members) in the *old group*. In the case of a search, SuperC creates a list of the files (or members) in the *search group*.

COMMAND field: At the top of the Selection List Menu is a COMMAND entry field. This field allows you to enter any of the selection menu commands: ADD, CANCEL, DOWN, LOCATE, RESET, SELECT, SELECT *, or UP.

For details, see "Selection Menu Commands" on page 309.

Note: You cannot enter CP or CMS commands in the COMMAND field.

The Selection List Menu (Comparison)

Figure 117 on page 307 shows an example of a Selection List Menu for a comparison of two file groups. The New File ID was entered as `TEMP * A` and the Old File ID was entered as `* TEMP A`.

.

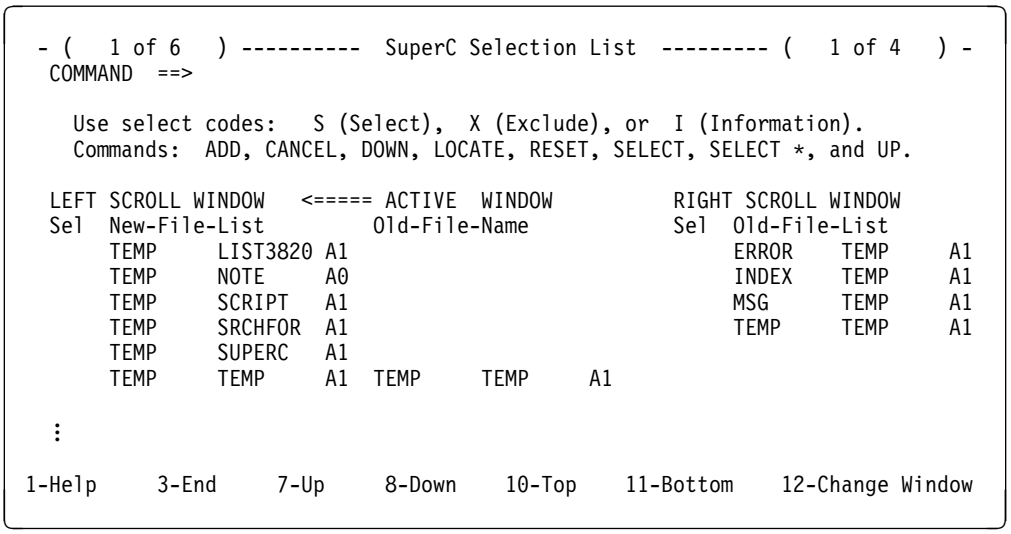

- (1 of 6) ---------- SuperC Selection List --------- (1 of 4) -
COMMAND ==>
Use select codes: S (Select), X (Exclude), or I (Information).
Commands: ADD, CANCEL, DOWN, LOCATE, RESET, SELECT, SELECT *, and UP.
LEFT SCROLL WINDOW <===== ACTIVE WINDOW RIGHT SCROLL WINDOW
Sel New-File-List Old-File-Name Sel Old-File-List
TEMP LIST3820 A1 ERROR TEMP A1
TEMP NOTE A0 INDEX TEMP A1
TEMP SCRIPT A1 MSG TEMP A1
TEMP SRCHFOR A1 TEMP TEMP A1
TEMP SUPERC A1
TEMP TEMP A1 TEMP TEMP A1
1-Help 3-End 7-Up 8-Down 10-Top 11-Bottom 12-Change Window

Figure 117. Example of a CMS Selection List Menu (File Group Comparison)

Comparison Scrollable Windows:

The LEFT SCROLL WINDOW consists of the following columns:

Sel A single-character selection code field. Allows you to enter one of the Selection Codes: S, X, I, or space (to unselect). For details, see page 309.

New-File-List Alphabetical list of files generated by SuperC as a result of the New File ID (containing an "*") entered on the Primary Comparison Menu.

New-Member (Not shown.) Alphabetical list of members generated by SuperC as a result of the Member Name (containing an "*") entered on the Primary Comparison Menu.

Old-File-Name This field must be filled with a valid ID from the *old* candidate list (right scroll window) to indicate with which *old* file the *new* file is to be compared.

You can manually pair *new* and *old* files by entering an "S" in the SEL column in the left scroll window and an "S" in the SEL column in the right scroll window. Then press Enter and the *old* file ID is entered alongside the *new* file ID.

Alternatively, you can enter the *old* file ID directly in the Old-File-Name field.

Note: Initially, SuperC automatically lists any items from the Old-File-List whose mask matches the mask from the New-File-List (see "How SuperC Pairs CMS Files and Members" on page 311).

Old-Member (Not shown.) This field is comparable to the Old-File-Name, but for MACLIB or TXTLIB members.

The RIGHT SCROLL WINDOW consists of the following columns:

Sel A single-character selection code field. Allows you to enter one of the Selection Codes: S, X, I, or space (to unselect). For details, see page 309.

Old-File-List Alphabetical list of files generated by SuperC as a result of the Old File ID (containing an "*") entered on the Primary Comparison Menu.

Old-Member-List (Not shown.) Alphabetical list of members generated by SuperC as a result of the Member Name (containing an "*") entered on the Primary Comparison Menu.

The arrow (<=====) on line 7 of the menu, indicates which "window" is active. (Use PF12 to toggle the active window between left and right.)

The Selection List Menu (Search)

Figure 118 shows an example of a Selection List Menu for a search of a file group. The New File ID was entered as `TEMP * A`.

```
- (   1 of 6   ) ----------  SuperC Selection List  -------------------------
 COMMAND  ==>

   Use  S (Select) or  X (Exclude) select codes.
   Commands:  ADD, CANCEL, DOWN, LOCATE, RESET, SELECT, SELECT *, and UP.

 Sel  Filename Filetype Fm Format Lrecl       Recs     Blocks   Date     Time
      TEMP     LIST3820 A1 V       3728         47          3  9/11/03 13:31:52
      TEMP     NOTE     A0 V         79        121          2  7/31/03 10:52:30
      TEMP     SCRIPT   A1 V         72        117          2  9/11/03 13:42:00
      TEMP     SRCHFOR  A1 V        111         14          1  9/15/03 14:55:44
      TEMP     SUPERC   A1 V        111        212          5  9/15/03 17:48:33
      TEMP     TEMP     A1 V         16          3          1  9/15/03 14:40:29

  ⋮

1-Help         3-End         7-Up         8-Down        10-Top       11-Bottom
```

Figure 118. Example of a CMS Selection List Menu (File Group Search)

Search Scrollable Window: The Selection List Menu for the SuperC Search has only one window.

When you are searching a group of files, the search scrollable window shows an alphabetical list of the files generated by SuperC as a result of the New File ID (containing an "*") entered on the Primary Comparison Menu.

When you are searching a MACLIB or TXTLIB, the search scrollable window shows an alphabetical list of the members generated by SuperC as a result of the Member Name (containing an "*") entered on the Primary Comparison Menu.

The search scrollable window consists of the following columns:

Sel A single-character selection code field. Allows you to enter one of the Selection Codes: S, X, I, or space (to unselect). For details, see page 309.

When searching *files*:

Filename Filetype Fm The search file ID.

Format F for Fixed; V for Variable.

Lrecl Record length.

Recs Number of records.

Blocks Number of blocks.

Date Date file was last modified.

Time Time file was last modified.

When searching MACLIB or TXTLIB *libraries* (not shown):

New-Member The member from the MACLIB or TXTLIB to be searched.

Selection Codes

The following selection codes can be entered in the selection ("SEL") field on the Selection List Menu:

S Select the item.

X Exclude the item from the list.

I Display information for this item. ("I" is not supported for search file selection as this information is already shown).

(space) "Unselect" the item.

Selection Menu Commands

The following commands may be entered on the Command line.

ADD Add additional files/members to the group in the active window. Previously matched files/members are unaltered. The add-mask may be a file/member name or may contain wildcard characters.

Format ADD *add_mask*

The abbreviation "A" is also acceptable.

Example
```
ADD AB* * A
```

This adds all those files on the A mini-disk whose file name starts with "AB"

BOTTOM Scrolls the active window to the last element in the list.

The abbreviations "BOT" and "B" are also acceptable.

CANCEL Cancel all selections. Return to the Primary Comparison Menu or the Primary Search Menu.

DOWN Scrolls the active window down one page (no operands) or the number of entries as indicated by the operand value. "DOWN" may also be indicated by using the PF8 key and a value operand as the command.

Format DOWN *value*

The abbreviation "D" is also acceptable.

LOCATE The LOCATE command scrolls to file (or member) you specify. If the specified name is not in the list, the data is scrolled to the file (or member) that precedes the specified name (in alphabetic sequence). LOCATE applies to the active window.

Format LOCATE *name*

Name is a fully-qualified CMS file ID for file list and a member name for member list.

The abbreviations “LOC” and “L” are also acceptable.

RESET The RESET command may be used to restore the selection list to its original status.

SELECT The SELECT command can be used to select a file (or member) that is contained within the selection list of the active window. If the specified name is not found, the list is scrolled to the file (or member) that precedes the specified name (in alphabetic sequence).

Format SELECT *name*

Name is a fully qualified CMS file ID for file list and a member name for member list.

The abbreviations “SEL” and “S” are also acceptable.

SELECT *

Selects all *new* entries (left window) that are paired to *old* entries (right window).

Format SELECT *

The abbreviations “SEL *” and “S *” are also acceptable.

TOP Scrolls the active window to the first element in the list.

UP Scrolls the active window up one page (no operands) or the number of entries as indicated by the operand value. “UP” may also be indicated by using the PF7 key and a value operand as the command.

Format UP *value*

File/Member Selection List PF Key Definitions

The PF keys which can be used while displaying the selection list are shown at the bottom of the screen. The following PF keys are defined on the Selection List Menu:

PF1 Help. PF1 displays detailed help information about the file selection list menu and commands.

PF3 Menu. PF3 exits this menu and executes the comparison (or search) if selections were made.

PF7 Up. PF 7 scrolls the list in the active window up one page or until the top of the list is reached. If a number is entered in the command field, PF7 7 scrolls up that number of lines (unless the top of the list is reached first).

PF8 Down. PF8 scrolls the list in the active window down one page or until the bottom of the list is reached. If a number is entered in the command field, PF8 scrolls down that many line (unless the bottom of the list is reached first).

PF10 Top. PF10 scrolls the active window to the top of the list.

PF11 Bottom. PF11 scrolls the active window to the bottom of the list.

PF12 Change-Window. PF12 toggles the active window from right to left or from left to right. PF12 is not applicable to the SuperC Search which has only one window.

How SuperC Pairs CMS Files and Members

It is important to understand how SuperC pairs files and members. Careless (or too frequent) use of the wildcard character ("*") can result in the wrong items being paired.

Pairing Files: SuperC pairs files in two stages. First, SuperC uses the LISTFILE command to collect the group of file names. Second, from this collection, SuperC pairs the *new* with the *old* based upon the asterisk-mask.

The asterisk-mask can be up to 16 characters long, 8 from the *fn* and 8 from the *ft*. Intervening spaces are discarded. The effects of this can be seen with the following example.

File Group ID	File ID	Asterisk-mask
* * A	XYZ ABC A	XYZABC
* ABC A	XYZ ABC A	XYZ
XYZ * A	XYZ ABC A	ABC
*YZ *BC A	XYZ ABC A	XA

The matching of pairs involves only the asterisk-masks. If the asterisk-masks are equal, the files are paired.

For example, if you had the following files on your A minidisk:

```
SUP      PLI   A
BKSUP    PLI   A
BKBKSUP  PLI   A
```

and specified `*SUP PLI A` on the New File ID field and `*BKSUP PLI A` on the Old File ID field, then SuperC generates the following collections using LISTFILE:

For *New* File	Asterisk-mask	For *Old* File	Asterisk-mask
BKBKSUP PLI A	"BKBK"	BKBKSUP PLI A	"BK"
BKSUP PLI A	"BK"	BKSUP PLI A	""
SUP PLI A	""		

For the *new* file BKBKSUP PLI A, SuperC substitutes "BKBK" for the asterisk-mask. Similarly, for *new* files BKSUP PLI A and SUP PLI A, SuperC substitutes "BK" and the null string respectively for the asterisk mask. For the *old* files BKBKSUP PLI A and BKSUP PLI A, SuperC substitutes "BK" and the null string respectively.

SuperC pairs the files when the asterisk-mask of the *new* file is the same as the asterisk-mask of the *old* file. Continuing with our example, SuperC pairs the following files:

```
New File                                  Old File
BKSUP   PLI  A          paired to         BKBKSUP  PLI  A
SUP     PLI  A          paired to         BKSUP    PLI  A
```

The first 2 files are paired together because the character string “BK” replaces the asterisk-mask of both files. The second 2 files are paired together because the null string is substituted for the asterisk-mask of both files.

Pairing Members: In a similar way to pairing files, pairing members from MACLIBs or TXTLIBs firstly involves the members being selected.

To be selected for the *new* list, the member must be a member of the *new* MACLIB or TXTLIB and have the same prefix as the prefix on the *new* Member field. To be selected for the *old* list, the member must be a member of the *old* MACLIB or TXTLIB and have the same prefix as the prefix on the *old* Member field.

For example:

```
New File ID ==> new  maclib  a             Member  ==> abc*
Old File ID ==> old  maclib  a             Member  ==> *
```

The *new* list consists of all members of NEW MACLIB A with a name having a prefix of “ABC” and the *old* list consists of all members of OLD MACLIB A.

Members are paired if the member name appears in both lists. For example, if the member name “ABCD” appears in both lists, the Member Selection List pairs the name.

CMS Files Used by SuperC

In addition to the files that you specify to be compared or searched, SuperC uses a number of CMS *work* files.

These *work* files are listed here:

SUPERC OLIST A
: The default file ID for the Options List file when invoking the SuperC Comparison on the CMS command line. (The Options List file holds a set of default options.)

  Note: The OLF keyword allows you to specify an alternative file ID for the Options List file.

SRCHFOR OLIST A
: The default file ID for the Options List file when invoking the SuperC Search on the CMS command line. (The Options List file holds a set of default options.)

  Note: The OLF keyword allows you to specify an alternative file ID for the Options List file.

SUPERC NAMES *
: Contains optional CMS command line information for the SuperC Comparison. (In most cases, the Options List file holds command line requirements.)

- If no equivalent options are specified in the CMS command line or the Options List file (user-specified or default), then options from the `LINE_DEF` tag of the SUPERC NAMES * file are used.
- Printer information for "WIDE" printing, if required, can be held in the SUPERC NAMES * file.

SUPERC SYSIN A
The default file ID for the process statements file when using the SuperC Comparison. A new SUPERC SYSIN A file (erasing the contents of an existing SUPERC SYSIN A file) is created when (comparison) process statements are selected using the Process Statements Entry Menu.

SRCHFOR SYSIN A
The default file ID for the process statements file when using the SuperC Search. A new SRCHFOR SYSIN A file (erasing the contents of an existing SRCHFOR SYSIN A file) is created when (search) process statements are selected using the Process Statements Entry Menu.

SUPERC $SYSIN$ A
A temporary control file used for passing CMS control line statement directives to SuperC.

Reasons for Differing Comparison Results

When comparing two sets of input date, it is possible that different compare types (FILE, LINE, WORD, and BYTE) gives slightly different results.

In order for SuperC to detect only the types of differences that are of interest to you, make sure that you are using the most appropriate compare type and, if necessary, the appropriate process options and process statements to select only the data that you actually want compared.

Here are some of the reasons why different compare types can produce different results:

- FILE and BYTE comparisons inspect the complete file (every byte) for differences. LINE and WORD comparisons use designated columns that are either specified by you or are within the default range of columns assigned by SuperC.

  For example, a FILE comparison of a file with fixed-length records of eighty bytes compares all columns (that is, all bytes), whereas a LINE comparison of the same file compares columns 1 to 72 (the default). Since the sequence number columns in the file are ignored in the LINE compare, the final results can be different. In this case, for consistent results, you should specify the LINE compare type and the NOSEQ process option.
- LINE comparisons "pad" the shorter records with spaces when comparing files with different record lengths. However, BYTE comparisons only "recognize" spaces when they are already present in the input file.
- For files with input line lengths <= 256, a LINE comparison is performed after padding the lines to the longest line length. Consequently two lines, originally of unequal length, compare equally only if the spaces at the end of the longer line coincide with the shorter line's space padding.

- For files with input line lengths > 256, a LINE comparison is performed on the lines without space padding. As a result, lines of unequal length are always a mismatch.
- Different compare types have different sensitivity to being resynchronized. Synchronization for a LINE comparison begins at the beginning of a line and ends at the end of a line. Synchronization for a WORD comparison begins anywhere on a line on any word boundary and ends at the end of a word. Synchronization for a BYTE comparison extends only one byte anywhere on a line.
- LINE comparisons detect lines that have been reformatted. However, reformatted lines have no effect on WORD comparisons as spaces and blank lines are ignored.
- Results may differ depending on which input file is specified as the “new” file and which is specified as the “old” file. The matching algorithm is sensitive to the largest matched *set* it finds between files. There may be occasions where more than one *set* of matched data meets this criteria. The rules for deciding which *set* to choose among the equals depends upon the contents of each file and which file was nominated as the “new” file.

Return Codes

SuperC displays the completion message at the top of the Primary Comparison Menu or at the top of the Primary Search Menu. The message is an interpretation of the following return codes.

Table 33 (Page 1 of 2). SuperC Return Codes

Code	Meaning
0	Normal completion.
	Comparison The input files are the same. No differences found.
	Search No matches found in the input file.
1	Normal completion.
	Comparison Differences were found in the input files.
	Search Matches found in the input file.
4	WARNING. Erroneous input was detected. Files were compared but results may not be as expected. Check listing for more information.
6	WARNING. *Old* file did not contain proper sequence numbers, or the sequence number intervals were not sufficiently large to contain insert activity (UPDCMS8 and UPDMVS8).
8	ERROR. Error on *old* input file. Files were NOT compared. Check listing for more details.
16	ERROR. Error on *new* or search source file. The operation was NOT performed. Check listing for more details.
20	ERROR. I/O error writing to update file, FILEDEF missing, or APNDUPD process option cancelled because of inconsistent file attributes.
24	ERROR. I/O error writing to the output listing file.
25	ERROR. The *old* output file attributes are not consistent with the new listing requirements. The APNDLST process option can not be accepted and the operation is immediately terminated.
26	ERROR. The output file caused a “disk full”condition. The output listing is incomplete.

Table 33 (Page 2 of 2). SuperC Return Codes

Code	Meaning
27	ERROR. The output file is a “read-only” disk. All I/O operations to the disk is suppressed.
28	ERROR. No data was compared because of invalid file names, no commonly named members of both input file groups, or one or both input files were empty.
32	ERROR. Insufficient storage was available for SuperC to execute. Refer to output listing for more details.
36	ERROR. VSE file would not open.
40	ERROR. VSE label information not available.
48	ERROR. VSE Librarian member not found.
52	ERROR. VSE VSAM Showcat failed.
56	ERROR. VSE device type not supported.
60	ERROR. Wrong length record read on tape input.

SuperC Messages

There are three levels of SuperC messages:

- *Informational* messages do not affect the return code and SuperC completes normally.
- *Warning* messages return a code of 4 to 7, processing is completed, but some user option/operation may not be completely performed.
- *Error* messages are accompanied with a return code of 8 or greater and the processing is prematurely terminated.

This section explains the SuperC message format and the messages you may receive.

Each of the messages issued by SuperC is of the form:

Message Format

ASMFnnns

where:

ASMF is the program identifier for SuperC

nnn represents a particular message number

s is the message severity level:

- **I** Informational message
- **W** Warning message
- **E** Error message

ASMF001I EMPTY COMPARE SET, INVALID NAMES, NO COMMON NAMED EMPTY FILES/DATA SETS, OR ZERO COMPARE AFTER FILTERED.

Explanation: No data has been found to be compared.

System Action: The SuperC run continues.

Programmer Response: Check that the file/member name(s) have been entered correctly. Also, check that the parameters for any select, focus/exclude options are correct.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF002I NO UPDATE FILE/DATA SET GENERATED FOR UPDCMS8 OR UPDMVS8 OPTIONS WHEN NO INPUT DIFFERENCES ARE FOUND.

Explanation: No differences in the input have been found. The update process option specified does not create an output update file in this situation.

System Action: The SuperC run continues.

Programmer Response: None.

ASMF003I THE COMPARISON OPERATION WAS EXECUTED UNDER STORAGE CONSTRAINTS THAT MAY AFFECT RESULTS/THROUGHPUT.

Explanation: Insufficient storage available for normal processing. Results are unpredictable. Output may be formatted incorrectly.

System Action: The SuperC run continues.

Programmer Response: Specify a larger region parameter on the JCL and resubmit the job.

ASMF004I LISTING LINES MAY BE TRUNCATED DUE TO LIMITING OUTPUT LINE WIDTH.

Explanation: The length of the data being printed is less than the length of one of the records. This is normal for a NARROW listing of 80 character records.

System Action: The SuperC run continues.

Programmer Response: The maximum listing length is 80 characters. If the data has records greater than 80, the part after the 80th character is not displayed. If the length of the data is between 56 and 80 characters, the WIDE option gives a side-by-side listing of 80 characters from each file.

See “Process Options” on page 226.

ASMF005I NO DATA SEARCHED INVALID NAME(S), EMPTY MEMBERS PROCESSED OR ZERO SEARCH SET AFTER INPUT FILTERING.

Explanation: No data has been found to be searched.

System Action: The SuperC run continues.

Programmer Response: Check that the file/member name(s) have been entered correctly. Also, check that the parameters for any SELECT, FOCUS/EXCLUDE process options are correct.

See “Process Options” on page 226.

ASMF006I UPDATE PROCESSING DETECTED SEQUENCE NUMBERING ERRORS.

Explanation: The sequence numbers on one or both input files have found to be incorrect.

System Action: The SuperC run continues.

Programmer Response: Check sequence numbering on input.

ASMF007I MOVED LINE FLAGGING ONLY VALID FOR FIRST 32K LINES PORTION OF COMPARE OPERATION PER DATA SET (OR FILE).

Explanation: Process option FMVLNS (Flag Moved Lines) restricted to a maximum of 32K “blocks” of moved lines.

System Action: The SuperC run continues.

Programmer Response: None.

ASMF009W GWCBL OPTION AND Y2DTONLY MUTUALLY EXCLUSIVE. GWCBL IS IGNORED.

Explanation: GWCBL and Y2DTONLY process options cannot be used together.

System Action: The SuperC run continues (without GWCBL process option).

Programmer Response: None.

ASMF010W *process-option* **PROCESS OPTION PARAMETER IS NOT A VALID PROCESS OPTION. IT IS IGNORED.**

Explanation: *process-option* is not a valid process option keyword and has been ignored.

System Action: The SuperC run continues.

Programmer Response: Check that the process option(s) have been entered correctly.

See “Process Options” on page 226.

ASMF011W *start-value* **SPECIFIED START VALUE GREATER THAN STOP VALUE. STOP VALUE CHANGED TO MAXIMUM VALUE.**

Explanation: When nominating a range, the start value for the range has been specified with value greater than the stop value for the range. SuperC has attempted to accommodate the range by extending the stop value to the maximum value for the line or file concerned.

System Action: The SuperC run continues.

Programmer Response: Check start and stop values for range(s).

ASMF012W **SRCHFOR STATEMENT(S) MISSING FOR SEARCH-FOR COMPARE TYPE REQUEST. ZERO LINES WILL BE INSPECTED.**

Explanation: SuperC expected 1 or more SRCHFOR process statements to be present (specifying the "string(s)" to be searched for) but none were found. No records searched.

System Action: The SuperC run continues.

Programmer Response: Check that "search string" is being supplied to SuperC correctly.

See (MVS) "Invoking the Search on MVS" on page 205, (CMS) "Invoking the Search on CMS using Menu Input" on page 207.and "Invoking the Search on CMS using Command Line Input" on page 215, (VSE) "Invoking the Search on VSE" on page 221.

ASMF013W **CERTAIN "DO NOT PROCESS" OPTIONS ARE REJECTED DUE TO LINE LENGTHS > 256. OPTIONS RESERVED FOR PROGRAM SOURCE DATA.**

Explanation: "Do not process" options are not allowed if line > 256 characters. These options are primarily for source text. The DPLINE process statement is allowed in these cases.

System Action: The SuperC run continues.

Programmer Response: Either use the DPLINE statement or modify the data before comparing.

ASMF014W **UPDATE OPTION CONFLICTS WITH "DO NOT PROCESS" OPTION SELECTION. "DO NOT PROCESS" OPTIONS IGNORED.**

Explanation: The update process option specified is incompatible with the "Do not process" (DP...) process option(s) specified.

System Action: The SuperC run continues.

Programmer Response: Check process options used.

See "Process Options" on page 226.

ASMF015W **UPDMVS8 AND UPDCMS8 PROCESS OPTIONS ARE ONLY ALLOWED WITH FIXED 80 RECORDS.**

System Action: The SuperC run continues. No update file is created.

Programmer Response: Check that the appropriate update process option is being used for the input file.

See "Process Options" on page 226.

ASMF016W **MOVE LINE DETECTION RESTRICTED TO LINES <= 256 LRECL. OPTION IS IGNORED.**

Explanation: Process option FMVLNS is restricted to lines <= 256 characters.

System Action: The SuperC run continues.

Programmer Response: None.

ASMF017W *file-name* **- SELECT MEMBER WAS NOT FOUND.**

Explanation: The member or file in the SELECT process statement could not be found.

System Action: The SuperC run continues.

Programmer Response: Check that the member/file name in the SELECT process statement is correct. Also, check that the "group" from which the member/file is to be selected has been specified correctly.

See "Process Statements" on page 236.

ASMF018W *file-name1:file-name2* **SELECT MEMBER-PAIR WAS NOT FOUND.**

Explanation: One or both of the members or files in the SELECT process statement could not be found.

System Action: The SuperC run continues.

Programmer Response: Check that both member/file names have been specified correctly.

See "Process Statements" on page 236.

ASMF019W **AGING PARAMETER IS INVALID**

Explanation: Aging parameter in NY2AGE/OY2AGE is not numeric. It should be a value between 1 and 999.

System Action: The SuperC run continues.

Programmer Response: Change NY2AGE/OY2AGE aging parameter to a valid value.

ASMF020W Y2DTONLY OPTION IGNORED AS THERE ARE NO VALID DATE DEFINITIONS.

Explanation: A Compare Dates Only (Y2DTONLY) process option has been specified but no dates have been defined by Date Definition (NY2C, NY2Z, NY2D, NY2P, OY2C, OY2Z, OY2D, OY2P) process statements.

System Action: The SuperC run continues.

Programmer Response: Use appropriate Date Definition process statements to define the date(s) to be compared.

See “Process Options” on page 226.

ASMF021W SYSIN ALTERNATE DDNAME PARAMETER INVALID.

Explanation: The name supplied is not a valid DD name or was not correctly supplied within parentheses.

System Action: The SuperC run continues without SYSIN process option.

Programmer Response: Ensure the rules for valid DD names are followed.

See “Process Options” on page 226.

ASMF022W *compare-type* **COMPARE TYPE AND THIS PROCESS STATEMENT ARE INCOMPATIBLE. STATEMENT IGNORED.**

Explanation: The compare type specified (FILE, LINE, WORD, or BYTE) is not valid for the process statement that has been specified.

System Action: The SuperC run continues.

Programmer Response: Change compare type to one that is valid for the process statement involved.

See “Process Statements” on page 236.

ASMF023W UNRECOGNIZED OR INVALID PROCESS STATEMENT KEYWORD.

Explanation: Keyword not valid for the process statement specified

System Action: The SuperC run continues.

Programmer Response: Check if the process statement involved requires a keyword. If so, ensure a valid keyword is used.

See “Process Statements” on page 236.

ASMF024W EXTRA DATA DETECTED AFTER NORMAL STATEMENT END. STATEMENT ACCEPTED WITH WARNING NOTIFICATION.

Explanation: Extraneous data or incorrect syntax.

System Action: The SuperC run continues.

Programmer Response: Check format of statement.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF025W INVALID PROCESS STATEMENT DATA-VALUE/OPERAND, EXTRA DATA OR EXCEEDS COLUMN 72. STMT/OPERAND IGNORED.

Explanation: Incorrect syntax for process statement.

System Action: The SuperC run continues.

Programmer Response: Check required syntax for process statement.

See “Process Statements” on page 236.

ASMF026W THE CMPBOFS STATEMENT AND UPDCNTL CONFLICT. STATEMENT IGNORED.

Explanation: Cannot use a CMPBOFS process statement with UPDCNTL process option.

System Action: The SuperC run continues. CMPBOFS process statement ignored.

Programmer Response: Change process option(s) or process statement(s) as necessary.

ASMF028W *statement-type* **STATEMENT CONFLICTS WITH SPECIFIED UPDATE OPTIONS. STATEMENT IGNORED.**

Explanation: The type of statement specified is not compatible with one or more of the update process options specified.

System Action: The SuperC run continues.

Programmer Response: See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF029W A SELECT PROCESS STATEMENT IS INVALID WITH SEQUENTIAL FILES/DATA SETS. STATEMENT IGNORED.

Explanation: SELECT process statements can only be used to select members/files from a “group.”

System Action: The SuperC run continues.

Programmer Response: See “Process Statements” on page 236.

ASMF030W THE SELECT STATEMENT HAS AN INVALID MEMBER NAME OR IMPROPER OPERAND FORMAT. STMT/MEMBER IGNORED.

Explanation: Incorrect content or syntax.

System Action: The SuperC run continues.

Programmer Response: Check that the member/file name(s) have been entered correctly in the SELECT process statement.

See “Process Statements” on page 236.

ASMF031W AN INVALID START COLUMN VALUE WAS SPECIFIED.

Explanation: Missing, non-numeric, or otherwise invalid “start column” parameter specified.

System Action: The SuperC run continues.

Programmer Response: Check that details have been entered correctly and in accordance with the required syntax.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF032W COLUMN VALUES MUST BE IN ASCENDING SEQUENCE. STATEMENT IGNORED.

Explanation: Column values not in ascending sequence or, possibly, statements out of sequence.

System Action: The SuperC run continues.

Programmer Response: Check that SuperC receives column numbers/ranges in ascending sequence such that a record can be scanned sequentially from “left to right.”

ASMF033W CMPCOLM RANGE STARTS WITH A VALUE EXCEEDING THE MAXIMUM PROCESSING LENGTH. STATEMENT TERMINATED.

Explanation: The “start_column” specified in the CMPCOLM process statement is greater than the logical record length of the file.

System Action: The SuperC run continues.

Programmer Response: Correct the column/range specified in the CMPCOLM process statement.

ASMF034W CMPCOLM STMT(S) HAS TOO MANY RANGES. ONLY FIRST 15 RANGES WILL BE USED.

Explanation: More than the permitted maximum of 15 ranges/individual columns specified for the CMPCOLM process statement. Extraneous information ignored.

System Action: The SuperC run continues.

Programmer Response: Limit ranges/individual columns to a maximum of 15 for each run of SuperC. Additional ranges/individual columns can be specified in a separate run.

ASMF035W INVALID CHANGE TEXT COMBINATION OF NEW TEXT > OLD TEXT AND LINE LENGTHS > 256 ATTRIBUTE.

Explanation: The length of the search text in a NCHGT or OCHGT process statement can not be greater than the length of the change text when a record is greater than 256 characters.

System Action: The SuperC run continues.

Programmer Response: Correct process statement.

ASMF036W SELECT STATEMENTS VALID ONLY WITH /PDS/MACLIBS/TXTLIBS OR “*” FILE NAMES. STATEMENT IGNORED.

Explanation: SELECT process statements can only be used to select members/files from a “group.”

System Action: The SuperC run continues.

Programmer Response:

See “Process Statements” on page 236.

ASMF037W DPLINEC MUST BE PRECEDED BY A VALID DPLINE/D PLINEC STATEMENT. STATEMENT REJECTED.

Explanation: The DPLINEC process statement is a continuation of the preceding DPLINE (or DPLINEC) statement and therefore must always be preceded by one of those statements.

System Action: The SuperC run continues.

Programmer Response: Ensure the first “Do not process” statement is a DPLINE followed, if necessary, by a DPLINEC statement containing “continuation” information.

See “Process Statements” on page 236.

ASMF038W SRCHFORC MUST BE PRECEDED BY A VALID SRCHFOR /SRCHFORC STATEMENT. STATEMENT REJECTED.

Explanation: The SRCHFORC process statement is a continuation of the preceding SRCHFOR (or SRCHFORC) statement and therefore must always be preceded by one of those statements.

System Action: The SuperC run continues.

Programmer Response: Ensure the first "search" statement is a SRCHFOR followed, if necessary, by a SRCHFORC statement containing "continuation" information.

See "Process Statements" on page 236.

ASMF039W ONLY ONE GROUP OF FILES OR MEMBERS MAY BE PROCESSED USING SELECTF STATEMENTS. STATEMENT REJECTED.

Explanation: SelectF does not allow multiple wildcard selection (except when used for *filemode*).

System Action: The SuperC run continues.

Programmer Response: Correct process statement.

See "Process Statements" on page 236.

ASMF040W SOME LINES OVERFLOW WITH CHANGE TEXT SUBSTITUTION. RESULTS MAY BE AFFECTED.

Explanation: Change text (NCHGT/OCHGT process statement) has a different length than search text. The result could run past the end of the record.

System Action: The SuperC run continues.

Programmer Response:

See "Process Statements" on page 236.

ASMF041W UPDLDEL OPTION INVALID DUE TO INCONSISTENT LRECL OR RECFM ATTRIBUTES.

Explanation: If input is fixed, then both files must be the same record length. The UPDLDEL option is ignored.

System Action: The SuperC run continues (without UPDLDEL process option).

Programmer Response: See "Process Options" on page 226.

ASMF042W NCHGT AND OCHGT MIXED DBCS PATTERNS MUST BE THE SAME LENGTH. STATEMENT REJECTED.

Explanation: The lengths of the search text and change text must be equal length in DBCS.

System Action: The SuperC run continues.

Programmer Response: Correct NCHGT or OCHGT process statement.

ASMF043W CMPCOLM NOT VALID FOR MIXED DATA AND SRCHCMP OR WORDCMP OPERATIONS. STATEMENT REJECTED.

Explanation: CMPCOLM process statement cannot be used with search or WORD compare type when the input contains a mixture of DBCS and non-DBCS data.

System Action: The SuperC run continues.

Programmer Response: Correct process statement or change to a line compare.

ASMF044W MIXING CMPLINE, CMPSECT, AND CMPBOFS STMTS IS NOT ALLOWED. STATEMENT REJECTED.

Explanation: Invalid combination of process statements.

System Action: The SuperC run continues.

Programmer Response: Use only one of these type of process statements at a time.

ASMF045W *statement-type* **STATEMENT(S) ONLY ALLOWED WITH SINGLE MEMBERS OR SEQUENTIAL FILES/DATA SETS. STATEMENT REJECTED.**

Explanation: An NTITLE, OTITLE or CMPSECT process statement has been used for a "group" of files or members. These statements are only valid for single members or files.

System Action: The SuperC run continues.

Programmer Response: Specify a single member/file.

ASMF046W VSE NEWDD/OLDDD PARAMETER IS INVALID.

Explanation: One of the NEWDD/OLDDD parameters is invalid.

System Action: The SuperC run continues.

Programmer Response: Check format of NEWDD/OLDDD statement.

See "Process Statements" on page 236.

ASMF047W VSE PARAMETER NAME LONGER THAN 8 CHARACTERS.

Explanation: One of the parameters on a NEWDD/OLDDD statement is too long.

System Action: The SuperC run continues.

Programmer Response: Correct NEWDD or OLDDD process statement.

See “Process Statements” on page 236.

ASMF048W VSE RECFORM VALUE MORE THAN 2 CHARACTERS.

Explanation: RECFORM in a NEWDD or OLDDD process statement can only be FU, FB, VU or VB.

System Action: The SuperC run continues.

Programmer Response: Correct NEWDD or OLDDD process statement.

See “Process Statements” on page 236.

ASMF049W VSE: MIXED MATCHING OF LIBRARIAN FILES AND SAM FILES NOT ALLOWED. NEWDD/OLDDD SET TO DEFAULT.

Explanation: Input files must be the same type. The statement which defined the Librarian is ignored and default attributes are assigned to the file concerned. Default file attributes used: fixed, unblocked, record and blocksize of 80.

System Action: The SuperC run continues.

Programmer Response: See “Process Statements” on page 236.

ASMF050W AGING ONLY ALLOWED ON ONE FILE. WILL ASSUME ONLY OLDDD IS TO BE AGED.

Explanation: NY2AGE and OY2AGE process statements are mutually exclusive. NY2AGE statement ignored.

System Action: The SuperC run continues.

Programmer Response: Check which file you want to “age” and use either the NY2AGE or OY2AGE accordingly.

See “Process Statements” on page 236.

ASMF051W CONFLICTING FOCUS/EXCLUDE STATEMENTS DEFINED.

Explanation: NEXCLUDE/OEXCLUDE process statements are mutually exclusive to NFOCUS/OFOCUS respectively if using the same operand keyword (ROWS or COLS).

System Action: The SuperC run continues.

Programmer Response: Check that the NEXCLUDE/OEXCLUDE and NFOCUS/OFOCUS process statements “exclude” and “focus” on the data you want without conflicting with each other.

See “Process Statements” on page 236.

ASMF052W WRONG DATE FORMAT IN NEW FILE

Explanation: Date definition format in NY2C/NY2Z/NY2D/NY2P statement is invalid. Date is ignored.

System Action: The SuperC run continues.

Programmer Response: Correct process statement.

See “Process Statements” on page 236.

ASMF053W WRONG DATE FORMAT IN OLD FILE

Explanation: Date definition format in OY2C/OY2Z/OY2D/OY2P statement is invalid. Date is ignored.

System Action: The SuperC run continues.

Programmer Response: Correct process statement.

See “Process Statements” on page 236.

ASMF054E “NEW” FILE/DATA SET NAME/MEMBER IS INVALID OR AN ERROR WAS ENCOUNTERED DURING OPEN. OPERATION TERMINATED.

Explanation: “New” input file could not be found or a problem was encountered during the open process.

System Action: The SuperC run terminates.

Programmer Response: Check that the “new” file name has been specified correctly

ASMF055E “OLD” FILE/DATA SET NAME/MEMBER IS INVALID OR AN ERROR WAS ENCOUNTERED DURING OPEN. OPERATION TERMINATED.

Explanation: “Old” input file could not be found or a problem was encountered during the open process.

System Action: The SuperC run terminates.

Programmer Response: Check that the “old” file name has been specified correctly

ASMF056E “SRH” FILE/DATA SET NAME/MEMBER IS INVALID OR AN ERROR WAS ENCOUNTERED DURING OPEN. OPERATION TERMINATED.

Explanation: New file could not be opened

System Action: The SuperC run terminates.

Programmer Response: Check that the dataset/file has been assigned correctly.

ASMF057E THE INPUT FILES/DATA SETS COULD NOT BE PROCESSED. BOTH MUST BE SEQUENTIAL OR A WHOLE PDS/MACLIB.

Explanation: Cannot compare a PDS/MACLIB/TXTLIB/Librarian with a sequential file/dataset.

System Action: The SuperC run terminates.

Programmer Response: Ensure input files are comparable.

ASMF058E MEMORY AVAILABLE WAS INSUFFICIENT. OPERATION TERMINATED.

Explanation: There was insufficient memory available for SuperC to run.

System Action: The SuperC run terminates.

Programmer Response: Increase amount of memory available.

ASMF059E A SYNAD ERROR INTERCEPT ON THE NEW-FILE/DATA SET IS AN I/O ERROR, CONCATENATION ORDERING OR ATTRIBUTE CONFLICT.

Explanation: New file/dataset I/O error.

System Action: The SuperC run terminates.

Programmer Response: Refer to your systems programmer.

ASMF060E A SYNAD ERROR INTERCEPT ON THE OLD-FILE/DATA SET IS AN I/O ERROR, CONCATENATION ORDERING OR ATTRIBUTE CONFLICT.

Explanation: Old file/dataset I/O error.

System Action: The SuperC run terminates.

Programmer Response: Refer to your systems programmer.

ASMF061E A SYNAD ERROR INTERCEPT ON THE UPD-FILE/DATA SET WAS DETECTED. THE OUTPUT MAY BE INCOMPLETE.

Explanation: Update file/dataset I/O error.

System Action: The SuperC run terminates.

Programmer Response: Refer to your systems programmer.

ASMF062E UPDATE FILE/DATA SET, DELDD, MISSING OR INCOMPATIBLE ATTRIBUTES/LRECL FOR PDS/MACLIB. UPDATE OPTIONS CANCELLED.

Explanation: Update/delta file requested but there is no assignment for it.

System Action: The SuperC run terminates.

Programmer Response: Refer to your systems programmer.

ASMF063E *member_name* **- SYNAD ERROR INTERCEPT OCCURRED PROCESSING NAMED MEMBER.**

Explanation: I/O error on processing member.

System Action: The SuperC run terminates.

Programmer Response: Refer to your systems programmer.

ASMF064E *data-set-name* **COULD NOT BE OPENED**

Explanation: Problem encountered when trying to open data set.

System Action: The SuperC run terminates.

Programmer Response: Correct either the *data-set-name* process statement or the *dataset-name* JCL statement.

ASMF065E LABEL INFORMATION NOT AVAILABLE FOR *data-set-name***.**

Explanation: Label details for data set missing.

System Action: The SuperC run terminates.

Programmer Response: Correct either the *data-set-name* process statement or the *data-set-name* JCL statement.

ASMF067E *data-set-name* **SHOWCAT FAILURE.**

Explanation: Error in accessing VSAM catalogue.

System Action: The SuperC run terminates.

Programmer Response: Make sure the *data-set-name* is assigned correctly.

ASMF068E *data-set-name* **DEVICE TYPE NOT SUPPORTED.**

Explanation: *data-set-name* is supported for disk and tape only.

System Action: The SuperC run terminates.

Programmer Response: Correct the *data-set-name* to ensure it is assigned to disk or tape.

ASMF069W LIBRARY MEMBER IN *data-set-name* NOT FOUND.

Explanation: Member could not be found in library.

System Action: The SuperC run continues (without this member).

Programmer Response: Inspect output listing for further details.

ASMF070W REQUEST FOR WIDE OPTION NOT SUPPORTED BY SYSLST. NARROW OPTION WILL BE SUBSTITUTED.

Explanation: The WIDE process option requires a printing device capable of printing lines up to 202 characters long. The 55-character side-by-side NARROW option has been used instead.

System Action: The SuperC run continues.

Programmer Response: Refer to your systems programmer.

ASMF071W SIDE BY SIDE LISTINGS NOT ALLOWED WHEN USING COLHEAD PROCESS STATEMENT.

Explanation: The NARROW process option cannot be used with the COLHEAD process statement.

System Action: The COLHEAD statements are accepted and the NARROW (side-by-side) process option is ignored. The SuperC run continues.

Programmer Response: Check that you are using the correct process options and statements.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF072W UPDATE PROCESS OPTIONS INCOMPATIBLE WITH Y2DTONLY PROCESS OPTION.

Explanation: Update process options cannot be used with the “Compare Dates Only” process option.

System Action: The UPD... process option is ignored. The SuperC run continues.

Programmer Response: Check that you are using the correct process options and statements.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF073W Y2PAST PROCESS STATEMENT SPECIFIED WITHOUT ANY DATE DEFINITION PROCESS STATEMENTS.

Explanation: A Y2PAST process statement has been used but there are no accompanying Date Definition process statements.

System Action: The Y2PAST process option is ignored. The SuperC run continues.

Programmer Response: Check that you are using the process statements correctly. Either the Y2PAST process statement should be removed, or one or more date definition process statements should be included.

See “Process Statements” on page 236.

ASMF074W FOCUS/EXCLUDE PROCESS STATEMENTS ARE IGNORED WHEN USING THE Y2DTONLY PROCESS OPTION.

Explanation: NFOCUS, OFOCUS, NEXCLUDE, and OEXCLUDE process statements have no effect when the Y2DTONLY process option is used.

System Action: The FOCUS/EXCLUDE process statements are ignored. The SuperC run continues.

Programmer Response: Check that you are using the correct process options and statements.

See “Process Options” on page 226 and “Process Statements” on page 236.

ASMF075I DATE DEFINITION PROCESS STATEMENTS ARE IGNORED WHEN USING THE COLHEAD PROCESS STATEMENT.

Explanation: Date Definition process statements cannot be used with the COLHEAD process statement. (The Date Definition statements generate their own *information line* for which column headings are not appropriate.)

System Action: The Date Definition process statements are ignored. The SuperC run continues.

Programmer Response: Check that you are using the correct process statements.

See “Process Statements” on page 236.

ASMF076I FOCUS/EXCLUDE OF ROWS USED FOR ONLY ONE FILE. ALL ROWS PROCESSED IN THE OTHER FILE.

Explanation: A “focus” (NFOCUS or OFOCUS) or an “exclude” (NEXCLUDE or OEXCLUDE) process statement has been specified for one file but not for the other file.

System Action: All rows (records) of the file for which no “focus” or “exclude” statement exists are included in the comparison process.

Programmer Response: Check that you are using the “focus” or “exclude” process statements correctly.

See “Process Statements” on page 236.

ASMF077E WRONG LENGTH RECORD IN *defined-input-file*. RUN ABORTED.

Explanation: The input tape file defined by the process statement *defined-input-file* (NEWDD or OLDDD) contains a record of the wrong length.

System Action: The SuperC run terminates.

Programmer Response: Check that the record format, block size, and maximum record size have been specified correctly on the NEWDD/OLDDD process statement.

See “Process Statements” on page 236.

ASMF079W FMSTOP OPTION ONLY VALID WITH FILE COMPARE OR SEARCH.

Explanation: The FMSTOP option is set for a compare that is not a FILE compare.

System Action: The FMSTOP option is ignored.

Programmer Response: Remove the FMSTOP option, or change the compare to a FILE compare.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie New York 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

CICS
BookMaster
DFSMS/MVS
IBM
IMS
MVS/DFP
MVS/ESA
OpenEdition
OS/390
OS/2
RETAIN
QMF
SP
System/370
VM/ESA
VSE/ESA
3090

Other company, product, and service names may be trademarks or service marks of others.

© Copyright IBM Corp. 1992, 2004

Glossary

This glossary defines terms and abbreviations that are used in this book. If you do not find the term you are looking for refer to the index, to the glossary of the appropriate high-level language (HLL) manual, or to the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

A

abend. Abnormal end of application.

accept. An SMP/E process that moves distributed code and programs to the distribution libraries.

activate. To make a program available for use.

addressing mode (AMODE). An attribute that refers to the address length that a routine is prepared to handle upon entry. Addresses may be 24 or 31 bits long.

address space. Domain of addresses that are accessible by an application.

AMODE. Addressing mode.

APAR. Authorized program analysis report.

authorized program analysis report (APAR). A request for correction of a problem caused by a defect in a current unaltered release of a program.

authorized program facility (APF). The authorized program facility (APF) is a facility that an installation manager uses to protect the system. In MVS, certain system functions, such as all or part of some SVCs, are sensitive; their use must be restricted to users who are authorized. An authorized program is one that executes in supervisor state, or with APF authorization.

auxiliary file. In CMS, a file that contains a list of file types of update files to be applied to a particular source file.

B

base. The core product, upon which features may be separately ordered and installed.

batch. Pertaining to activity involving little or no user action. Contrast with *interactive*.

byte. The basic unit of storage addressability, usually with a length of 8 bits.

C

cataloged procedure. A set of control statements placed in a library and retrievable by name.

CBIPO. Custom-Built Installation Process Offering.

CBPDO. Custom-Built Product Delivery Offering.

CE. IBM customer engineer.

CLIST. TSO command list.

CMS. Conversational monitor system.

compiler options. Keywords that can be specified to control certain aspects of compilation. Compiler options can control the nature of the load module generated by the compiler, the types of printed output to be produced, the efficient use of the compiler, and the destination of error messages.

component. (1) Software that is part of a functional unit. (2) A set of modules that performs a major function within a system.

condition code. A code that reflects the result of a previous input/output, arithmetic, or logical operation.

control block. A storage area used by a computer program to hold control information.

control file. In CMS, a file that contains records that identify the updates to be applied and the macrolibraries, if any, needed to assemble a particular source program.

control program (CP). A computer program designed to schedule and to supervise the execution of programs of a computer system.

control section (CSECT). The part of a program specified by the programmer to be a relocatable unit, all elements of which are to be loaded into adjoining main storage locations.

control statement. (1) In programming languages, a statement that alters the continuous sequential execution of statements; a control statement can be a conditional statement, such as IF, or an imperative statement, such as STOP. (2) In JCL, a statement in a job that identifies the job or describes its requirements to the operating system.

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program

© Copyright IBM Corp. 1992, 2004

development capabilities, and operates only under the control of the VM/370 control program.

corrective maintenance. Maintenance performed specifically to overcome existing problems.

CP command. In VM, a command by which a terminal user controls his or her virtual machine. The VM/370 control program commands are called CP commands.

CPPL. Command processor parameter list.

CP privilege class. In VM, one or more classes assigned to a virtual machine user in the user's VM directory entry; each privilege class allows access to a logical subset of the CP commands.

CSI. Consolidated software inventory data set.See *SMPCSI*.

CSECT. Control section.

cumulative service tape. A tape sent with a new function order, containing all current PTFs for that function.

Custom-Built Installation Process Offering (CBIPO). A CBIPO is a tape that has been specially prepared with the products (at the appropriate release levels) requested by the customer. A CBIPO simplifies installing various products together.

Custom-Built Product Delivery Offering (CBPDO). A CBPDO is a tape that has been specially prepared for installing a particular product and the related service requested by the customer. A CBPDO simplifies installing a product and the service for it.

D

data definition name (DDNAME). The logical name of a file within an application. The DDNAME provides the means for the logical file to be connected to the physical file.

data set. Under MVS, a named collection of related data records that is stored and retrieved by an assigned name. Equivalent to a CMS *file*.

data set name (dsname). The data set name on the DD statement in the JCL or the dsname operand of the TSO ALLOC command.

DBCS. Double-byte character set.

DDDEF. Dynamic data definition.

DDNAME. Data definition name.

default. A value that is used when no alternative is specified.

DD statement. In MVS, connects the logical name of a file and the physical name of the file.

DELTA disk. In VM, the virtual disk that contains program temporary fixes (PTFs) that have been installed but not merged.

disassembler. A program that accepts object code as input, and produces assembler language source statements and a pseudo-listing as output.

disassembly. The process of converting object code into assembler language source statements and a pseudo-listing.

distribution libraries. IBM-supplied partitioned data sets on tape containing one or more components that the user restores to disk for subsequent inclusion in a new system.

distribution medium. The medium on which software is distributed to the user; for example, 9-track magnetic tape, tape cartridge.

distribution zone. In SMP/E, a group of VSAM records that describe the SYSMODs and elements in the distribution libraries.

DLBL. VSE only. Disk Label information; JCL statement.

double-byte character set (DBCS). A collection of characters represented by a 2-byte code.

driving system. The system used to install the program. Contrast with target system.

dsname. Data set name.

dynamic data definition (DDDEF). The process of defining a data set and allocating auxiliary storage space for it while, rather than before, a job step executes.

dynamic storage. Storage acquired as needed at run time. Contrast with *static storage*.

E

ECMODE. Extended control mode.

executable program. (1) A program that has been link-edited and therefore can run in a processor. (2) The set of machine language instructions that constitute the output of the compilation of a source program.

Extended control mode (ECMODE). A mode in which all features of a System/370 computing system, including dynamic address translation, are operational.

Extended Service Option (ESO). A service option that gives a customer all the new fixes for problems in IBM licensed programs that operate under that customer's operating system.

F

feature. A part of an IBM product that may be ordered separately by a customer.

feature number. A four-digit code used by IBM to process hardware and software orders.

file. A named collection of related data records that is stored and retrieved by an assigned name. Equivalent to an MVS *data set*.

FILEDEF. File definition statement.

file definition statement (FILEDEF). In CMS, connects the logical name of a file and the physical name of a file.

fix. A correction of an error in a program, usually a temporary correction or bypass of defective code.

FMID. Function modification identifier.

function. A routine that is invoked by coding its name in an expression. The routine passes a result back to the invoker through the routine name.

function modification identifier (FMID). The value used to distinguish separate parts of a product. A product tape or cartridge has at least one FMID.

H

HLASM. The High Level Assembler.

I

IBM customer engineer (CE). An IBM service representative who performs maintenance services for IBM hardware.

IBM program support representative (PSR). An IBM service representative who performs maintenance services for IBM software at a centralized IBM location.

IBM service representative. An individual in IBM who performs maintenance services for IBM products or systems.

IBM Software Distribution (ISD). The IBM department responsible for software distribution.

IBM Support Center. The IBM department responsible for software service.

IBM systems engineer (SE). An IBM service representative who performs maintenance services for IBM software in the field.

initial program load (IPL). (1) The initialization procedure that causes an operating system to commence operation. (2) The process by which a configuration image is loaded into storage, as at the beginning of a work day or after a system malfunction or as a means to access updated parts of the system. (3) The process of loading system programs and preparing a system to run jobs.

inline. Sequential execution of instructions, without branching to routines, subroutines, or other programs.

IPL. Initial program load.

interactive. Pertaining to a program or system that alternately accepts input and responds. In an interactive system, a constant dialog exists between user and system. Contrast with *batch*.

ISD. IBM Software Distribution.

J

JCL. Job control language.

JCLIN data. The JCL statements associated with the ++JCLIN statement or saved in the SMPJCLIN data set. They are used by SMP/E to update the target zone when the SYSMOD is applied. Optionally, SMP/E can use the JCLIN data to update the distribution zone when the SYSMOD is accepted.

JES. Job Entry Subsystem

Job Entry Subsystem. A system facility for spooling, job queueing, and managing the scheduler work area.

job control language (JCL). A sequence of commands used to identify a job to an operating system and to describe a job's requirements.

job step. You enter a program into the operating system as a job step. A job step consists of the job control statements that request and control execution of a program and request the resources needed to run the program. A job step is identified by an EXEC statement. The job step can also contain data needed by the program. The operating system distinguishes job control statements from data by the contents of the record.

L

library. A collection of functions, subroutines, or other data.

link pack area (LPA). In MVS, an area of main storage containing reenterable routines from system libraries. Their presence in main storage saves loading time when a reenterable routine is needed.

linkage editor. A program that resolves cross-references between separately assembled object modules and then assigns final addresses to create a single relocatable load module. The linkage editor then stores the load module in a program library in main storage.

link-edit. To create a loadable computer program by means of a linkage editor.

load module. An application or routine in a form suitable for execution. The application or routine has been compiled and link-edited; that is, address constants have been resolved.

logical saved segment. A portion of a physical saved segment that CMS can manipulate. Each logical saved segment can contain different types of program objects, such as modules, text files, execs, callable services libraries, language repositories, user-defined objects, or a single minidisk directory. A system segment identification file (SYSTEM SEGID) associates a logical saved segment to the physical saved segment in which it resides. See *physical saved segment* and *saved segment*.

LPA. Link pack area.

M

MCS. Modification control statement

minidisk. In VM, all, or a logical subdivision of, a physical disk storage device that has its own address, consecutive storage space for data, and an index or description of stored data so that the data can be accessed. Synonymous with virtual disk.

module. A language construct that consists of procedures or data declarations and can interact with other such constructs.

MSHP. Maintain system history program.

MVS. Multiple Virtual Storage operating system.

N

Named Saved System. A copy of an operating system that a user has named and saved in a file. The user can load the operating system by its name, which is more efficient than loading it by device number.

national language support (NLS). Translation requirements affecting parts of licensed programs; for example, translation of message text and conversion of symbols specific to countries.

NLS. National language support.

nonexecutable components. Components of a product that cannot be run.

non reentrant. A program that cannot be shared by multiple users.

nonreenterable. See *non reentrant*.

NSS. named saved system

O

object code. Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

object deck. Synonymous with *object module, text deck*.

object module. A portion of an object program suitable as input to a linkage editor. Synonymous with *text deck, object deck*.

online. (1) Pertaining to a user's ability to interact with a computer. (2) Pertaining to a user's access to a computer via a terminal.

operating system. Software that controls the running of programs; in addition, an operating system may provide services such as resource allocation, scheduling, input/output control, and data management.

P

parameter. Data items that are received by a routine.

phase. VSE only. A link edited program.

partition. A fixed-size division of storage.

physical saved segment. One or more pages of storage that have been named and retained on a CP-owned volume (DASD). When created, it can be loaded within a virtual machine's address space or outside a virtual machine's address space. Multiple

users can load the same copy. A physical saved segment can contain one or more logical saved segments. A system segment identification file (SYSTEM SEGID) associates a physical saved segment to its logical saved segments. See *logical saved segment* and *saved segment*.

preventive maintenance. Maintenance performed specifically to prevent problems from occurring.

preventive service planning (PSP). The online repository of program temporary fixes (PTFs) and other service information. This information could affect installation.

procedure. A named block of code that can be invoked, usually via a call.

procedure library (PROCLIB). A program library in direct access storage with job definitions. The reader/interpreter can be directed to read and interpret a particular job definition by an execute statement in the input stream.

PROCLIB. Procedure library.

program level. The modification, release, version, and fix level of a product.

program number. The seven-digit code (in the format *xxxx-xxx*) used by IBM to identify each program product.

program temporary fix (PTF). A temporary solution or bypass of a problem diagnosed by IBM as resulting from a defect in a current unaltered release of the program.

PSP. Preventive service planning.

PSR. IBM program support representative.

PTF. Program temporary fix.

Q

qualifier. A modifier that makes a name unique.

R

reentrant. The attribute of a routine or application that allows more than one user to share a single copy of a load module.

reenterable. See *reentrant*

relative file tape (RELFILE tape). A standard label tape made up of two or more files. It contains a file of the MCSs for one or more function SYSMODs and one or more relative files containing unloaded source data sets and unloaded, link-edited object data sets at the distribution library level. A relative file tape is one way of packaging SYSMODs, and is typically used for function SYSMODs.

relative files (RELFILEs). Files containing modification text and JCL input data associated with a SYSMOD.

RELFILEs. Relative files

RELFILE tape. Relative file tape

relocatable load module. Under CMS, a combination of object modules having cross references resolved and prepared for loading into storage for execution.

residence mode (RMODE). The attribute of a load module that specifies whether the module, when loaded, must reside below the 16MB virtual storage line or may reside anywhere in virtual storage.

resident modules. A module that remains in a particular area of storage.

return code. A code produced by a routine to indicate its success. It can be used to influence the execution of succeeding instructions.

RIM. Related installation materials

RMODE. Residence mode.

run. To cause a program, utility, or other machine function to be performed.

S

save area. Area of main storage in which contents of registers are saved.

SBCS. Single-byte character set.

service level. The modification level, release, version, and fix level of a program. The service level incorporates PTFs if there are any.

saved segment. A segment of storage that has been saved and assigned a name. Saved segments can be physical saved segments that CP recognizes or logical saved segments that CMS recognizes. The segments can be loaded and shared among virtual machines, which helps use real storage more efficiently, or a private, nonshared copy can be loaded into a virtual machine. See *logical saved segment* and *physical saved segment*.

shared segment. In VM, a feature of a saved system that allows one or more segments of reenterable code

in real storage to be shared among many virtual machines.

shared storage. An area of storage that is the same for each virtual address space. Because it is the same space for all users, information stored there can be shared and does not have to be loaded in the user region.

severity code. A part of run-time messages that indicates the severity of the error condition (1, 2, 3, or 4).

single-byte character set (SBCS). A collection of characters represented by a 1-byte code.

SMPCSI. The SMP/E data set that contains information about the structure of a user's system as well as information needed to install the operating system on a user's system. The SMPCSI DD statement refers specifically to the CSI that contains the global zone. This is also called the master CSI.

softcopy. One or more files that can be electronically distributed, manipulated, and printed by a user.

software inventory disk. In VM, the disk where the system level inventory files reside.

source code. The input to a compiler or assembler, written in a source language.

source program. A set of instructions written in a programming language that must be translated to machine language before the program can be run.

SREL. System release identifier

statement. In programming languages, a language construct that represents a step in a sequence of actions or a set of declarations.

SUBSET. The value that specifies the function modifier (FMID) for a product level. It further specifies an entry in RETAIN* for a product level.

subsystem. A secondary or subordinate system, or programming support, usually capable of operating independently of or asynchronously with a controlling system. Examples are CICS and IMS.

SVA. Shared virtual area.

syntax. The rules governing the structure of a programming language and the construction of a statement in a programming language.

SYSMOD. system modification.

SYSMOD ID. system modification identifier.

system abend. An abend caused by the operating system's inability to process a routine; can be caused by errors in the logic of the source routine.

T

target disk. In VM, the disk to which a program is installed.

target libraries. In SMP/E, a collection of data sets in which the various parts of an operating system are stored. These data sets are sometimes called system libraries.

target zone. In SMP/E, a collection of VSAM records describing the target system macros, modules, assemblies, load modules, source modules, and libraries copied from DLIBs during system generation, and the system modifications (SYSMODs) applied to the target system.

text deck. Synonym for *object module, object deck*.

time sharing option/extended (TSO/E). An option on the operating system; for System/370, the option provides interactive time sharing from remote terminals.

TSO/E. Time sharing option/extended.

U

UCLIN. In SMP/E, the command used to initiate changes to SMP/E data sets. Actual changes are made by subsequent UCL statements.

UPGRADE. An alphanumeric identifier that specifies a product level.

user exit. A routine that takes control at a specific point in an application.

USERMOD. User modification.

user modification (USERMOD). A change to product code that the customer initiates.

V

virtual machine (VM). (1) A functional simulation of a computer and its associated devices. Each virtual machine is controlled by a suitable operating system. (2) In VM, a functional equivalent of either a System/370 computing system or a System/370-Extended Architecture computing system.

VMFINS. An installation aid supplied as part of VMSES/E to make installation on VM consistent.

VM Serviceability Enhancements Staged/Extended (VMSES/E). A program product for installing and maintaining products on VM.

VMSES/E. VM Serviceability Enhancements Staged/Extended.

VOLSER. Volume serial number.

volume. (1) A certain portion of data, together with its data carrier, that can be handled conveniently as a unit. (2) A data carrier mounted and demounted as a unit; for example, a reel of magnetic tape, a disk pack.

volume label. An area on a standard label tape used to identify the tape volume and its owner. This area is the first 80 bytes and contains VOL 1 in the first four positions.

volume serial number (VOLSER). A number in a volume label assigned when a volume is prepared for use in a system.

VSAM. Virtual storage access method. A high-performance mass storage access method. Three types of data organization are available: entry sequenced data sets (ESDS), key sequenced data sets (KSDS), and relative record data sets (RRDS).

word. A space-delimited character string.

Bibliography

High Level Assembler Publications

HLASM General Information, GC26-4943

HLASM Installation and Customization Guide, SC26-3494

HLASM Language Reference, SC26-4940

HLASM Licensed Program Specifications, GC26-4944

HLASM Programmer's Guide, SC26-4941

Toolkit Feature Publications

HLASM Toolkit Feature User's Guide, GC26-8710

HLASM Toolkit Feature Debug Reference Summary, GC26-8712

HLASM Toolkit Feature Interactive Debug Facility User's Guide, GC26-8709

HLASM Toolkit Feature Installation and Customization Guide, GC26-8711

Related Publications (Architecture)

Enterprise Systems Architecture/390 Principles of Operation, SA22-7201

System/370 Enterprise Systems Architecture Principles of Operation, SA22-7200

System/370 Principles of Operation, GA22-7000

System/370 Extended Architecture Principles of Operation, SA22-7085

z/Architecture Principles of Operation, SA22-7832

Related Publications for MVS

z/OS:

z/OS MVS JCL Reference, SA22-7597

z/OS MVS JCL User's Guide, SA22-7598

z/OS MVS Programming: Assembler Services Guide, SA22-7605

z/OS MVS Programming: Assembler Services Reference, Volumes 1 and 2, SA22-7606, SA22-7607

z/OS MVS Programming: Authorized Assembler Services Guide, SA22-7608

z/OS MVS Programming: Authorized Assembler Services Reference, Volumes 1 - 4, SA22-7609 - SA22-7612

z/OS MVS Program Management: User's Guide and Reference, SA22-7643

z/OS MVS System Codes, SA22-7626

z/OS MVS System Commands, SA22-7627

z/OS MVS System Messages, Volumes 1 - 10, SA22-7631 - SA22-7640

z/OS Communications Server: SNA Programming, SC31-8829

OS/390:

OS/390 MVS JCL Reference, GC28-1757

OS/390 MVS JCL User's Guide, GC28-1758

OS/390 MVS Programming: Assembler Services Reference, GC28-1910

OS/390 MVS System Codes, GC28-1780

OS/390 MVS System Commands, GC28-1781

OS/390 MVS System Messages, Volumes 1 - 5, GC28-1784 - GC27-1788

OpenEdition:

MVS/ESA OpenEdition MVS User's Guide, SC23-3013

z/OS UNIX System Services User's Guide, SA22-7801

MVS/DFP™:

MVS/DFP Version 3.3: Utilities, SC26-4559

MVS/DFP Version 3.3: Linkage Editor and Loader, SC26-4564

DFSMS/MVS®:

z/OS DFSMS Program Management, SC27-1130

z/OS DFSMSdfp Utilities, SC26-7414

TSO/E (z/OS):

z/OS TSO/E Command Reference, SA22-7782

TSO/E (OS/390):

OS/390 TSO/E Command Reference, SC28-1869

SMP/E (z/OS):

SMP/E Messages, Codes and Diagnosis, GA22-7770

SMP/E Reference, SA22-7772

SMP/E User's Guide, SA22-7773

SMP/E (OS/390):

© Copyright IBM Corp. 1992, 2004

SMP/E Messages and Codes, SC28-1738

SMP/E Reference, SC28-1806

SMP/E User's Guide, SC28-1740

Related Publications for VM

z/VM:

z/VM CMS Application Development Guide, SC24-6002

z/VM CMS Application Development Guide for Assembler, SC24-6003

z/VM CMS User's Guide, SC24-5968

z/VM CMS Command Reference, SC24-5969

z/VM CMS File Pool Planning, Administration, and Operation, SC24-6058

z/VM System Messages and Codes - CMS, GC24-6031

z/VM XEDIT User's Guide, SC24-5972

z/VM XEDIT Command and Macro Reference, SC24-5973

z/VM CP Command and Utility Reference, SC24-5967

z/VM Planning and Administration, SC24-5995

z/VM Service Guide, GC24-5946

z/VM VMSES/E Introduction and Reference, GC24-5994

Related Publications for VSE

VSE/ESA Administration, SC33-6705

VSE/ESA Guide to System Functions, SC33-6711

VSE/ESA Installation, SC33-6704

VSE/ESA Planning, SC33-6703

VSE/ESA System Control Statements, SC33-6713

VSE/ESA Messages and Codes, Vols.1 - 3, SC33-6796, SC33-6798, SC33-6799

Index

Special Characters

240
< 93
* 93
* SuperC process statement 240
*-wildcard
  See SuperC asterisk-wildcard
** 93
%-wildcard
  See SuperC percent-wildcard
> 93

A

ADATA assembly option 64
ADATA file names 67
  listed 67
ADATA files
  closing 74
  creating 64
  downloading 64
  listed 91
  opening 67
  removing 74
  sample 66
  working with 67
adding context 81
analyzing 62
  assembler language programs 62
arcs
  appearance 93
  colors 93
  double-clicking 89
  return 83
area box
  scrolling 89
  zooming 86
ASIS field (SuperC primary search menu)
  See SuperC search, primary menu fields
ASMLEAVE macro 12, 21, 25
ASMMREL macro 12
ASMPUT
  closing 92
  introduction to 62
  slide show demo 66
ASMPUT analysis messages
  hiding 71
  purpose 71
  showing 71
ASMPUT Control Flow Graph window
  closing 67
ASMPUT Control Flow Graph window *(continued)*
  described 93
  icons 95
  opening 67
  options 95
ASMPUT file list area
  contents 67
  position 91
ASMPUT finding
  See finding in ASMPUT
ASMPUT icons
  Collapse 75
  Collapse to Context 81
  Control Flow Graph window 95
  Expand 75
  Help 92
  Main window 92
  Open file 67
  Redo 82
  Refresh 82
  Show Context 81
  Show Graph 67
  Show Notebook 92
  Show Overview 85
  Show Return Arcs 83
  Show Zoom Slider 86
  Zoom In 86
  Zoom In Rectangle 86
  Zoom Out 86
  Zoom Out Rectangle 86
ASMPUT Main window
  described 91
  file list area 91
  file list area contents 67
  function 67
  icons 92
  information notebook 92
  opening ADATA files 67
  options 92
  source code area 92
  viewing source code 68
ASMPUT messages 97
ASMPUT options
  Center On 89
  Collapse All Layers 75
  Collapse in Context 75
  Collapse Layer 75
  Collapse to Context 81
  Control Flow Graph window 95
  Expand in Context 75
  Expand to Window 75
  Find 71

© Copyright IBM Corp. 1992, 2004

ASMPUT options *(continued)*
Find Next 71
Find Next Diagnostic/Message 71
Help Topics 92
Main window 92
Mark 84
Open 67
Redo Layout 82
Refresh 82
Remove 74
Remove All 74
Remove Context 81
Scroll to Source 89
Scroll to Target 89
Show Analysis Messages 71
Show Assembly Diagnostics 70
Show Context 81
Show Expanded Lines 69
Show Graph 67
Show Info Notebook 92
Show Overview 85
Show Return Arcs 83
Show Zoom Slider 86
Unmark 84
Unmark All 84
Zoom In 86
Zoom In On 86
Zoom In Rectangle 86
Zoom Out 86
Zoom Out From 86
Zoom Out Rectangle 86
ASMPUT shortcut keys
described 92
finding 71
finding next 71
ASMPUT tabs
HLASM files 92
Job Id 92
Libraries 92
Options 92
Statistics 92
ASMPUT What's This help 97
ASMPUT windows
Control Flow Graph 93
described 91
Main 91
Main file list area 91
Main information notebook 92
Main source code area 92
Overview 95
ASMXREF
ASMXREF EXEC statement, VSE 123
ASMXREP EXEC statement, VSE 124
CMS 115
CMS EXEC 115
control statements 125
ASMXREF *(continued)*
DLBL statement, VSE 123
EXEC ASMXREF statement, CMS 119
EXEC ASMXREF statement, MVS 112
EXEC ASMXRPT statement 112
EXEC ASMXRPT statement, CMS 119
invoking 115
JCL requirements, MVS 109
JCL requirements, VSE 120
messages 158
MVS batch 109
MVS procedures 113
MVS sample JCL 109
options 132
SYSIN DD statement 112
understanding reports 135
using 106
VSE 120
ASMXREF control files
CMS 115
MVS 112
VSE 120
ASMXREF control statements
* 125
EXCLUDE 127
INCLUDE 126
LIBRARY 125, 126, 128
PARM 127
REPORT 128
ASMXREF examples
CMS EXEC ASMXREF 119
EXCLUDE control statement 127
INCLUDE control statement 127
LIB parameter to LIBRARY control statement 126
LIBRARY control statement 126
MASK on token control statement 130
MVS JCL 109
REPORT control statement 128
VSE JCL 121
ASMXREF options 132
DUP 132
MSGLEVEL 132
NODUP 132
PAGELEN 132
ASMXREF token statements 129
EXC 129
INC 129
MASK 130
NODEFLT 130
NOSEP 131
ASMXREP
EXEC 119
EXEC statement 112, 124
JCL requirements 112, 124
JCL requirements, MVS 109
JCL requirements, VSE 124

ASMXREP *(continued)*
MVS sample JCL 109
options 135
understanding CR report 136
understanding LOC report 139
understanding MWU report 147
understanding SOR 148
understanding SWU report 150
understanding TSP 155
understanding TWU report 154
using 106
VSE sample JCL 121
ASMXREP options 135
ASMXRPT MVS procedure 109
ASMXSCAN CMS EXEC 119
ASMXSCAN MVS procedure 109
assembler instructions 92
assembler language programs
analyzing 62
assembly diagnostics (ASMPUT)
hiding 70
purpose 70
showing 70
assembly options
ADATA 64
GOFF 64
XOBJECT 64
auto display pgm field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
auto display pgm field (SuperC primary search menu)
See SuperC search, primary menu fields

B

backward indexing 27
block comment 141, 142
branch relative on condition instructions 13
branching to the ENDDO 23

C

C family references 137
CAPS field (SuperC primary search menu)
See SuperC search, primary menu fields
CASE macro 12, 34
CASE structured programming macro set 34
CASENTRY macro 12, 34
CC (SuperC process stmt directive)
See ?
Center On option 89
centering 89
change flags 140
multiplication factor 143
rules for counting 144
standard format 143
Change-Flag Descriptor 140, 141
field definition 141
implicit flags 141
process class codes 141
standard format 141
changed source instructions (CSI) 139
changing font properties 68
CHGNV SuperC process statement 238
closing
ADATA files 74
ASMPUT 92
Control Flow Graph window 67
Overview window 85
CMPBOFS SuperC process statement 241
CMPCOLM SuperC process statement 241
CMPCOLMN SuperC process statement 241
CMPCOLMO SuperC process statement 241
CMPLINE SuperC process statement 242
CMPSECT SuperC process statement 244
CMS
ASMXREF CMS EXEC 115
ASMXREF invoking with EXECs 115
ASMXRPT EXEC 119
Disassembler requirements 44
CMS example for Disassembler 45
CMS EXECs
ASMXREF 115
invoking ASMXREF in CMS 115
CMS XRFLANG file 118
COLHEAD SuperC process statement 249
Collapse All Layers option 75
Collapse icon 75
Collapse in Context option 75
Collapse Layer option 75
Collapse to Context icon 81
Collapse to Context option 81
collapsing layers 75
color coding of source code 92
colors
arcs 93
nodes 93
command field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
command field (SuperC primary search menu)
See SuperC search, primary menu fields
comment Disassembler statement 52
output description
SYSPRINT - SYSLST 54
SYSPUNCH - SYSPCH 53
comments 92
comments, full line
definition 140
compare type field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
component name format 141

context
adding 81
collapsing in 75
collapsing to 81
described 81
removing 81
showing 81
control file
ASMXREF for CMS EXEC 115
ASMXREF for MVS batch 109
ASMXREF for VSE batch 123
control file in CMS 116
Control Flow (CF) report 136
control flow graph
described 62
interacting with source code 89
working with 74
control flow graph area
cyan 93
described 93
gray 93
green 93
magenta 93
target 93
yellow 93
Control Flow Graph window
See ASMPUT Control Flow Graph window
Control statements 124, 125
ASMXREF 124
control statements for Disassembler 48
COPY Disassembler statement 52
COPY segments 92
copyright
observing, on disassembly 42
counting 27
counting comments 140
creating ADATA files 64
Cross-Reference Facility 106
CSECT Disassembler statement 48
cyan node 93

D

DATA only Disassembler statement 49
default ASMXREF options file 117
default token list 132, 133
defaults (Control Flow Graph window)
restoring 69
defaults (Main window)
restoring 69
Disassembler
CMS Example 45
Comment statement 52
control statements 48
COPY statement 52
DATA-only statement 49
Disassembler *(continued)*
Disassembler options 46
disassembling a module for the first time 52
DS-area statement 50
DSECT definition 50
DSECT header statement format 50
INSTR-only statement 49
invoking 42
module-CSECT statement 48
MVS JCL example 42
options on CMS 46
options on MVS 44
options on VSE 47
output description 53, 54
PARM field (MVS) 44
PARM field (VSE) 47
ULABL statement 51
USING statement 51
VSE JCL example 47
Disassembler, using the 41
display output option (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
DO loop terminator generation 22
DO macro 12, 21
DO structured programming macro indexing group 22
DO structured programming macro set 21
documentation
High Level Assembler 337
DOEXIT macro 12, 21
double-clicking
an arc 89
downloading ADATA files 64
DPLINE SuperC process statement 250
DPLINEC SuperC process statement 250
DS area Disassembler statement 50
DSECT Disassembler definition 50
DSPL
See display output option
DUP option in ASMXREF 132

E

ELSE macro 12, 14
ELSEIF macro 12, 20
ENDCASE macro 12, 34
ENDDO macro 12, 21
ENDIF macro 12, 14
ENDLOOP macro 12, 31
ENDSEL macro 12, 37
ENDSRCH macro 12, 31
EXC keyword for ASMXREF 127, 129
EXEC statement
ASMXREF MVS EXEC statement 112
ASMXREF VSE EXEC statement 123
ASMXREP VSE EXEC statement 124

EXEC statement *(continued)*
ASMXRPT MVS EXEC statement 112
MVS ASMXRPT EXEC statement 112
sample MVS JCL 109
sample VSE JCL 121
EXITIF macro 12, 31
Expand All Layers option 75
Expand icon 75
Expand in Context option 75
Expand Layer option 75
Expand to Window option 75
expanded lines
hiding all 69
hiding for one line 69
showing all 69
showing for one line 69
expanding layers 75
explicit specification 26

F

file information 72
viewing 72
file list area
See ASMPUT file list area
File Selection List
See SuperC CMS file selection list
file transfer to PC 149
Find Next Diagnostic/Message option 71
Find Next option 71
Find option 71
finding in ASMPUT
in source code 71
next diagnostic or message 71
shortcut keys for 71
fn ft fm
See ?
format notation, description v—vii
format option 135
FORMAT option in ASMXREP 135
forward indexing 28

G

generic matching rules 131
GOFF assembly option 64
gray node 93
green node 93
grouping flags 141

H

hiding
analysis messages 71
assembly diagnostics 70
expanded lines 69
hiding *(continued)*
return arcs 83
zoom slider 86
highlighted source code 92
HLASM Files information
viewing 73
HLASM Files tab 92

I

icons
See ASMPUT icons
IF macro 12, 14
IF structured programming macro option A 15
IF structured programming macro option B 16
IF structured programming macro option C 16
IF structured programming macro option D 17
IF structured programming macro set 14
IF structured programming macros with Boolean operators 18
implicit flag 143
INC keyword for ASMXREF 129
infinite loop 23
information notebook 92
INSTR only Disassembler statement 49
intellectual property rights 42
introduction to ASMPUT 62
invoking ASMXREF 108
general 108
invoking the Disassembler 42
CMS requirements 44
MVS requirements (JCL) 42
VSE requirements 46
ITBSIZE 127
ITERATE macro 12, 21, 23

J

JCL
(MVS) Disassembler 42
(VSE) Disassembler 46
ASMXREF VSE JCL requirements 123
ASMXREP VSE JCL requirements 124
ASMXRPT MVS EXEC statement 112
MVS ASMXREF EXEC statement 112
MVS ASMXREF invoking 109
MVS ASMXREF requirements 109
MVS ASMXREF sample JCL 109
MVS ASMXREF SYSIN DD statement 112
MVS ASMXREP requirements 109
MVS SuperC EXEC statement 181, 206
MVS SuperC invoking 180, 205
MVS SuperC requirements 180, 205
VSE ASMXREF invoking 120
VSE ASMXREF requirements 120
VSE ASMXREF sample JCL 121

JCL *(continued)*
VSE ASMXREP EXEC statement 124
VSE ASMXREP requirements 124
VSE EXEC ASMXREF statement 123
VSE SuperC invoking 199, 221
VSE SuperC requirements 199, 221
VSE SuperC sample JCL 199, 222
XRFLANG DLBL statement 123
XRFTOKN DLBL statement 123
JCL (VSE) example for Disassembler 47
Job Id information
viewing 73
Job Id tab 92

K

keyboard shortcuts 92

L

labels 92
language file 132
default token segment 133
language segment 134
layers
collapsing 75
expanding 75
LC (SuperC process stmt directive)
See ?
leaving a nested DO 25
Libraries tab 92
LIBRARY control statement in ASMXREF
ASMXREF 106, 125
Library information
viewing 73
license inquiry 327
Lines Of Code (LOC) report 139
lines of OO code (LOOC) report 145
linked node 62
listing file examples
DLMDUP listing 279
group FILE compare 281
LOCS listing 282
NARROW listing 278
side-by-side listing 278, 280
WIDE listing 280
listing file ID field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
listing file ID field (SuperC primary search menu)
See SuperC search, primary menu fields
listing type field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
LNCT SuperC process statement 254
LOC per Class section
sample report 147
LOC per Object section
sample report 146
LOGSIZE 127
LOOC
sample report 146
LPSFV SuperC process statement 254
LSTCOLM SuperC process statement 254
LT (SuperC process stmt directive)
See ?

M

machine instructions 92
machine instructions, publications 337
macro calls 92
Macro Where Used (MWU) report 147
macros 13, 14, 21, 31, 34, 37
ASMMREL macro 13
case macro set 34
DO macro set 21
IF macro 14
search macro set 31
select macro set 37
magenta node 93
Main window
See ASMPUT Main window
manuals
High Level Assembler 337
Mark option 84
marked node 93
marking nodes 84
MASK keyword for ASMXREF 130, 131
matching rules in ASMXREF 131
maximum zoom 86
member field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
member field (SuperC primary search menu)
See SuperC search, primary menu fields
Member Selection List
See SuperC CMS file selection list
message level option
ASMXREF in CMS EXEC 115
ASMXREF options 132
MVS ASMXREF EXEC statement 112
VSE ASMXREF EXEC statement 123
message list 159
messages
ASMXREF 158
severity codes 158
messages Disassembler
CMS 55
general 57
minimum zoom 86
module CSECT statement 48
MSGLEVEL option in ASMXREF 132

multiplication factor 143
MVS
  ASMXREF EXEC statement 112
  ASMXREF invoking with JCL 109
  ASMXREF sample JCL 109
  ASMXREF SYSIN DD statement 112
  ASMXREP JCL requirements 109
  ASMXRPT EXEC statement 112
  Disassembler JCL requirements 42
  SuperC EXEC statement 181, 206
  SuperC invoking with JCL 180, 205
MVS procedure
  ASMXRPT 109
  ASMXSCAN 109
MVS publications 337
MWUSIZE 127

N

name prefixes 93
NCHGT SuperC process statement 238
new file ID field (SuperC primary comparison menu)
  See SuperC comparison, primary menu fields
NEWDD SuperC process statement 245, 246
NEXCLUDE SuperC process statement 252
next diagnostic or message
  finding 71
NFOCUS SuperC process statement 253
node colors 93
NODEFLT keyword for ASMXREF 130
nodes
  See nodes in ASMPUT control flow graph
nodes in ASMPUT control flow graph
  color 93
  described 63
  double-clicking 75
  marked 93
  marking 84
  name prefixes 93
  program entry 75
  secondary entry 75
  selected 93
  selecting 89
  source 93
  three-dimensional 93
  two-dimensional 93
  unmarking 84
  unresolved 75
  yellow 84
NODUP option in ASMXREF 132
NOSEP keyword for ASMXREF 131
notation, description v—vii
NTITLE SuperC process statement 262
NY2AGE SuperC process statement 263
NY2C SuperC process statement 263
NY2D SuperC process statement 263
NY2P SuperC process statement 263
NY2Z SuperC process statement 263

O

OCHGT SuperC process statement 238
OEXCLUDE SuperC process statement 252
OFOCUS SuperC process statement 253
old file ID field (SuperC primary comparison menu)
  See SuperC comparison, primary menu fields
OLDDD SuperC process statement 245, 246
online help 66, 96
OOSIZE 127
Open file icon 67
Open option 67
opening
  ADATA files in ASMPUT 67
  Control Flow Graph window 67
  Overview window 85
operands 92
options
  See ASMPUT options
options file 117
options for ASMXREF 132
options for ASMXREP 135
Options information
  viewing 73
Options tab 92
ORELSE macro 12, 31
other resources 66
OTHRWISE macro 12, 37
OTITLE SuperC process statement 262
Overview window
  closing 85
  described 95
  opening 85
OY2AGE SuperC process statement 263
OY2C SuperC process statement 263
OY2D SuperC process statement 263
OY2P SuperC process statement 263
OY2Z SuperC process statement 263

P

PAGELEN option in ASMXREF 132
PARM control statement in ASMXREF 127
PARM field (MVS) for Disassembler 44
PARM field (VSE) for Disassembler 47
PARM option on LIBRARY control statement 127
PL family references 137
pop-up menu
  Control Flow Graph window 95
  Main window 92
primary entry point 75

process options field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
process options field (SuperC primary search menu)
See SuperC search, primary menu fields
process statements
See SuperC comparison process statements
See SuperC search process statements
process statements ID field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
process statements ID field (SuperC primary search menu)
See SuperC search, primary menu fields
product name format 141
program entry nodes 75
program entry point 75, 93
publications
High Level Assembler 337
HLASM Toolkit 337
machine instructions 337
MVS 337
VM 338
VSE 338

R

railroad track format, how to read v—vii
rc
See SuperC Return Codes
redo 82
Redo icon 82
Redo Layout option 82
refresh 82
Refresh icon 82
Refresh option 82
register initialization 29
remarks 92
definition 140
Remove All option 74
Remove Context option 81
Remove option 74
removed context 74
removing
ADATA files 74
context 81
REPORT control statement in ASMXREF
ASMXREF 128
REPORT option in ASMXREP 112
reports in ASMXREF
ASMXREP VSE EXEC 124
ASMXRPT CMS EXEC 119
ASMXRPT MVS EXEC 112
Control Flow report 136
Lines of Code report 139
list of available 106
reports in ASMXREF *(continued)*
Macro Where Used report 147
options 135
Spreadsheet Oriented report 148
Symbol Where Used 150
Tagged Source Program 155
Token Where Used report 154
understanding 135
reports, understanding 135
resizing windows 91
restoring defaults (Control Flow Graph window) 69
restoring defaults (Main window) 69
restoring fonts (Main window) 69
return arcs
hiding 83
showing 83
REVREF SuperC process statement 255
REXX references 138
RR (SuperC process stmt directive)
See ?

S

sample ADATA files 66
scan rules for ASMXREF 131
scroll bars 89
Scroll to Source option 89
Scroll to Target option 89
scrolling 89
search file ID field (SuperC primary search menu)
See SuperC search, primary menu fields
SEARCH macro 31
search option directives
ERASRC0 267
NOIMSG 267
NONAMES 267
NOOLF 267
PRINT 267
search process options
ALLMEMS 228
ANYC 228
APNDLST 228
COBOL 229
DPACMT 229
DPADCMT 229
DPBLKCL 230
DPCBCMT 230
DPCPCMT 230
DPFTCMT 230
DPMACMT 230
DPPLCMT 230
DPPSCMT 230
IDPFX 231, 289
LMCSFC 231
LMTO 231, 285—286, 290—291
LNFMTO 231

search process options *(continued)*
LONGLN 231
LPSF 232, 292
LTO 232, 291
MIXED 232
NOPRTCC 232
NOSEQ 232
NOSUMS 232
SEQ 233
XREF 235, 286—288, 290, 291
SEARCH structured programming macro set 31
secondary entry nodes 75
secondary entry point 75, 93
SELECT macro 12, 37
SELECT structured programming macro set 37
SELECT SuperC process statement 258, 259, 260
SELECTF SuperC process statement 257
selecting a node 89
selection list (SuperC primary search menu)
See SuperC search, primary menu fields
sequence numbers 92
shipped source instructions (SSI) 139, 146
shortcut keys
See ASMPUT shortcut keys
Show Analysis Messages option 71
Show Assembly Diagnostics option 70
Show Context icon 81
Show Context option 81
Show Expanded Lines option 69
Show Graph icon 67
Show Graph option 67
Show Info Notebook option 92
Show Notebook icon 92
Show Overview icon 85
Show Overview option 85
Show Return Arcs icon 83
Show Return Arcs option 83
Show Zoom Slider icon 86
Show Zoom Slider option 86
showing in ASMPUT
analysis messages 71
assembly diagnostics 70
expanded lines 69
return arcs 83
zoom slider 86
slide show demo 66
SLIST SuperC process statement 261
sort order option 135
source code
changing font 68
finding text in 71
highlight 92
interacting with control flow graph 89
viewing 68
source code area 92
color coding 92
source list file in CMS 117
source node 93
Spreadsheet Oriented (SOR) report 148
SRCH (SuperC process stmt directive)
See SuperC search process statement directives
SRCHFOR SuperC process statement 255
SRCHFORC SuperC process statement 255
stacked items vi
standard change flags 141, 142, 143
standard flags 139
starting 64
Statistics information
viewing 73
Statistics tab 92
STRTSRCH macro 12, 31
structured programming macros 13, 14, 15, 16, 17, 18, 20, 21, 22, 23, 25, 26, 27, 28, 29, 31, 34, 37
ASMMREL macro 13
backward indexing 27
branch relative on condition instructions 13
branching to the ENDDO 23
CASE macro set 34
counting 27
DO indexing group 22
DO loop terminator generation 22
DO macro set 21
ELSEIF macro 20
explicit specification 26
forward indexing 28
IF macro option A 15
IF macro option B 16
IF macro option C 16
IF macro option D 17
IF macro set 14
IF macros with Boolean operators 18
infinite loop 23
leaving a nested DO 25
register initialization 29
SEARCH macro set 31
SELECT macro set 37
UNTIL keyword 29
WHILE keyword 29
SuperC asterisk-wildcard 184
SuperC CMS command line option directive 266
SuperC CMS command line statement option directives 267
SuperC CMS file selection list
ADD 307, 309
BOTTOM 309
CANCEL 307, 309
COMMAND 306
DOWN 307, 309
File/Member Selection List 311
File/Member Selection List commands 306, 307
left scroll window 307
new Sel column 307
new-file-list 307

SuperC CMS file selection list *(continued)*
left scroll window *(continued)*
new-member-list 307
old-file-name 307
old-member-name 307
LOCATE 307, 309
PF key definitions 310—311
RESET 307, 310
right scroll window 307
old Sel field 308
old-file-list 308
old-member-list 308
SELECT 307, 310
SELECT * 307, 310
selection list 305—311
TOP 310
UP 307, 310
SuperC CMS files used 312
SuperC Comparison
EXEC SuperC statement, MVS 181
JCL requirements, MVS 180
JCL requirements, VSE 199
MVS batch 180
MVS sample JCL 180
VSE 199
VSE sample JCL 199
SuperC comparison listing
| (change bar) 273
change bar (|) 273
column title line 271
D (deleted line) 272
DC (delete compose) 272
DEL= (delete TYPE code) 273
delete compose (DC) 272
delete moved (DM) 272
delete replace (DR) 272
deleted line (I) 272
DLMDUP listing example 279
DM (delete moved) 272
DMR= (delete-move-reformat TYPE code) 273
DMV= (delete-move TYPE code) 273
DR (delete replace) 272
FILE compare of file groups 281
I (inserted line) 272
IC (insert compose) 272
id column 271
ID column (listing file) 271
IM (insert moved) 272
IMR= (insert-move-reformat TYPE code) 273
IMV= (insert-move TYPE code) 273
INS= (insert TYPE code) 273
insert compose (IC) 272
insert moved (IM) 272
inserted line (I) 272
LEN column 271
LEN column (listing file) 271
SuperC comparison listing *(continued)*
LOCS listing example 282
MAT= (match TYPE code) 273
match compose (MC) 272
MC (match compose) 272
member summary section 270
N-LN# 271
N-LN# (listing file) 271
NARROW listing example 278
O-LN# 271
O-LN# (listing file) 271
overall summary section 270
page headings
compare date 270
compare time 270
new file ID 271
old file ID 271
page number 271
printer control character 270
program date 270
program ID 270
program version 270
reformat new (RN) 272
reformat old (RO) 272
RFM= (reformat TYPE code) 273
RN (reformat new) 272
RO (reformat old) 272
RPL= (replace TYPE code) 273
scale 271
section title line 271
side-by-side listing example 278, 280
source line column 271
SOURCE LINE column (listing file) 271
TYPE column 271
TYPE column (listing file) 271
WIDE listing example 280
SuperC comparison process statement directives
CC 268
LC 268
LT 268
RR 269
SuperC comparison process statements
240
* 240
CHNGV 238
CMPBOFS 241
CMPCOLM 241—242
CMPCOLMN 241—242
CMPCOLMO 241—242
CMPLINE 242—243
CMPSECT 244—245
COLHEAD 249—250
DPLINE 250—252
DPLINEC 250—252
LNCT 254
LSTCOLM 254

SuperC comparison process statements *(continued)*
NCHGT 238—240
NEWDD 245—246—249
NEXCLUDE 252—253
NFOCUS 253—254
NTITLE 262, 271
NY2AGE 263
NY2C 263—266
NY2D 263—266
NY2P 263—266
NY2Z 263—266
OCHGT 238—240
OEXCLUDE 252—253
OFOCUS 253—254
OLDDD 245—246—249
OTITLE 262, 271
OY2AGE 263
OY2C 263—266
OY2D 263—266
OY2P 263—266
OY2Z 263—266
REVREF 255
SELECT 258—259—260—261
SELECTF 185, 257—258, 281
SLIST 261
UPDDD 245—246—249
WORKSIZE 262
Y2PAST 266
SuperC comparison type
BYTE 185, 186
FILE 186
LINE 176, 186, 271
WORD 176, 186, 283
SuperC comparison, on CMS command line input
FILELIST 198
OLF 198
options list file 198
process statements menu 196
PROMPT 196
SUPERC NAMES * 198
SuperC comparison, on CMS menu input
* (selection list) 185
asterisk-wildcard 184
auto display pgm 191
BROWSE 191
compare type 186
COND (display output option) 190, 191
display output 191
display output option 190—191
EPDF 191
execute and quit 191
file ID (new file ID) 184
file ID (old file ID) 184
file_id (listing file id) 187
file_id (process statements id) 190
file_id (process stmts id) 189
SuperC comparison, on CMS menu input *(continued)*
file_id (update file id) 190
File/Member selection list 185
help 191
hex dump 186
listing file ID 187
listing type 186—187
MACLIB/TXTLIB member 185
new file ID 184
NO (display output option) 190
NO (selection list) 185
old file ID 184
percent-wildcard 185
PF key definitions (SuperC) 191
print 191
proc opts 191
process option menus 191
process statements 191
process statements file 189—190
process statements menu 191
quit 191
selection list 185
SuperC Primary Menu 184
SuperC Process Statements Entry Menu 190
UPD (display output option) 191
update file ID 190
WIDE listing 192
WIDE print menu 191
XEDIT 191
YES (display output option) 190, 191
SuperC comparison, on MVS 180
SuperC comparison, on VSE 199
SuperC comparison, primary menu fields
auto display pgm 184, 191, 214—215
command 184, 208
compare type 184, 186
display output option 184, 190—191, 214
listing file ID 184, 187, 213—214
listing type 184, 186—187
member 184, 185, 209
new file ID 184
old file ID 184
process options 188—189, 212—213
process statements ID 184, 189—190, 214
selection list 185, 211
update file ID 184, 190
SuperC introduction 174
applications 177
find match example 175
SuperC listing type
CHNG 186, 187
DELTA 186, 187, 271
LONG 185, 186, 187
NOLIST 187
OVSUM 186, 187

SuperC listings 269
SuperC messages 316
SuperC option directives
  ERASRC0 267
  MENU 267
  NOIMSG 267
  NONAMES 267
  NOOLF 267
  PRINT 267
SuperC pairing of CMS files and members 311
SuperC percent-wildcard
  CMS SuperC Command Line 192
    syntax 192
SuperC process options
  ALLMEMS 228
  ANYC 228
  APNDLST 228
  APNDUPD 228
  CKPACKL 229
  CNPML 229
  COBOL 229
  COVSUM 229
  DLMDUP 229, 279
  DLREFM 229
  DPACMT 229
  DPADCMT 229
  DPBLKCL 230
  DPCBCMT 230
  DPCPCMT 230
  DPFTCMT 230
  DPMACMT 230
  DPPLCMT 230
  DPPSCMT 230
  FMSTOP 230
  FMVLNS 230, 272
  GWCBL 231, 273
  LOCS 231, 282
  LONGLN 231
  NARROW 232, 278
  NOPRTCC 232, 270, 283
  NOSEQ 232
  NOSUMS 232
  REFMOVR 232
  SDUPM 233
  SEQ 233
  SYSIN 233
  UPDCMS8 190, 233, 295—296
  UPDCNTL 190, 233, 296—299
  UPDLDEL 190, 233, 299
  UPDMVS8 190, 234, 300
  UPDPDEL 190, 234, 301
  UPDREV 176, 190, 234, 293—294
  UPDREV2 190, 234, 294—295
  UPDSEQ0 190, 235, 301—302
  UPDSUMO 190, 235, 302—305
  WIDE 235, 280
SuperC process options *(continued)*
  XWDCMP 186, 236
  Y2DTONLY 236
SuperC process statements
  + (DPLINE operand) 251
  + (SRCHFOR operand) 256
  +start_column (DPLINE operand) 251
  +start_column (SRCHFOR operand) 256
  B (COLHEAD keyword) 249
  BTM (CMPBOFS keyword) 241
  BTM (CMPLINE keyword) 243
  BTM (CMPSECT keyword) 244
  C (COLHEAD keyword) 249
  D (COLHEAD keyword) 249
  end_col (CMPSECT operand) 245
  end_column (CMPCOLM operand) 242
  end_position (NEXCLUDE operand) 253
  end_position (NFOCUS operand) 253
  end_position (OEXCLUDE operand) 253
  end_position (OFOCUS operand) 253
  fixed (Y2PAST operand) 266
  hex_offset (CMPBOFS operand) 241
  last_start_column (CMPLINE operand) 243
  last_start_column (DPLINE operand) 251
  last_start_column (LSTCOLM operand) 254
  last_start_column (NCHGT operand) 239
  last_start_column (OCHGT operand) 239
  last_start_column (SRCHFOR operand) 256
  line number (CMPLINE operand) 243
  NBTM 241
  NBTM (CMPBOFS keyword) 241
  NBTM (CMPLINE keyword) 243
  NBTM (CMPSECT keyword) 244
  new_file_ID (SELECTF operand) 257
  new_member (SELECT operand) 259, 260
  new_name (SELECT operand) 258
  NTOP 241
  NTOP (CMPBOFS keyword) 241
  NTOP (CMPLINE keyword) 243
  NTOP (CMPSECT keyword) 244
  number (CHNGV operand) 238
  number (LNCT operand) 254
  number (LPSFV operand) 254
  OBTM 241
  OBTM (CMPBOFS keyword) 241
  OBTM (CMPLINE keyword) 243
  OBTM (CMPSECT keyword) 244
  OFF (SLIST operand) 261
  old_file_ID (SELECTF operand) 257
  old_member (SELECT operand) 259, 261
  old_name (SELECT operand) 258
  ON (SLIST operand) 261
  OTOP 241
  OTOP (CMPBOFS keyword) 241
  OTOP (CMPLINE keyword) 243
  OTOP (CMPSECT keyword) 244

SuperC process statements *(continued)*
output_string (NCHGT operand) 239
output_string (OCHGT operand) 239
P (COLHEAD keyword) 249
P (SRCHFOR operand) 256
RCVAL=number (REVREF operand) 255
REFID=name (REVREF operand) 255
S (SRCHFOR operand) 256
search_file_ID (SELECTF operand) 257
search_member (SELECT operand) 260, 261
search_name (SELECT operand) 258
search_string (CMPLINE operand) 243
search_string (CMPSECT operand) 245
search_string (NCHGT operand) 239
search_string (OCHGT operand) 239
section ID (CMPSECT operand) 244
sliding (Y2PAST operand) 266
start_column (CMPCOLM operand) 242
start_column (CMPLINE operand) 243
start_column (CMPSECT operand) 245
start_column (DPLINE operand) 251
start_column (LSTCOLM operand) 254
start_column (NCHGT operand) 239
start_column (OCHGT operand) 239
start_column (SRCHFOR operand) 256
start_position (NEXCLUDE operand) 252
start_position (NFOCUS operand) 253
start_position (OEXCLUDE operand) 252
start_position (OFOCUS operand) 253
string (SRCHFOR operand) 256
title_name (NTITLE operand) 262
title_name (OTITLE operand) 262
TOP (CMPBOFS keyword) 241
TOP (CMPLINE keyword) 243
TOP (CMPSECT keyword) 244
W (SRCHFOR operand) 256
Z (COLHEAD keyword) 249
SuperC reasons for differing comparison results 313
SuperC return codes
descriptions 314—315
empty input file error 315
error 314—315
error return codes 314—315
file attributes (inconsistent) 314
inconsistent file attributes 314
insufficient storage error 315
invalid sequence numbers 314
listing file error (disk full) error 314
listing file error (read only) error 315
listing file I/O error 314
no common members/files to compare 315
no data to compare error 315
normal completion 314
normal completion return codes 314
storage (insufficient) error 315
update file error (read only) error 315
SuperC return codes *(continued)*
update file I/O error 314
warning 314
warning return codes 314
SuperC Search
EXEC SuperC statement, MVS 206
JCL requirements, MVS 205
JCL requirements, VSE 221
MVS batch 205
MVS sample JCL 205
VSE 221
VSE sample JCL 222
SuperC search listing
page headings 283
compare date 284
compare time 284
page number 284
printer control character 283
program date 283
program ID 283
program version 283
SuperC search process statement directives
CC 268
LC 268
LT 268
SRCH(string) 221
SuperC search process statements
240
* 240
CMPCOLM 241—242
CMPLINE 242—243
CMPSECT 244—245
COLHEAD 249—250
DPLINE 250—252
DPLINEC 250—252
LNCT 254
LPSFV 254—255
LSTCOLM 254
NCHGT 238—240
NTITLE 262
SELECT 258—259—260—261
SELECTF 257—258
SLIST 261
SRCHFOR 255—257
SRCHFORC 255—257
SuperC search, on CMS command line input
& ("AND") 221
| ("OR") 221
invoking on CMS, command line input 215
OLF 221
options list file 221
process statements menu 219
PROMPT 219
SUPERC NAMES * 221
SuperC search, on CMS menu input
* (selection list) 211

SuperC search, on CMS menu input *(continued)*
asterisk-wildcard 208
auto display pgm 214—215
BROWSE 214
COND (display output option) 214
display output 214
EPDF 214
file ID (new file ID) 208
file ID (old file ID) 208
file_id (listing file id) 213—214
file_id (process statements id) 214
File/Member selection list 211
listing file ID 213—214
MACLIB/TXTLIB member 209
new file ID 208
NO (selection list) 211
old file ID 208
PF key definitions (search) 215
process statements file 214
selection list 211
SuperC Primary Search Menu 208
SuperC Process Statements Entry Menu 214
UPD (display output option) 214
XEDIT 214
YES (display output option) 214
SuperC search, on MVS 205
SuperC search, on VSE 221
SuperC search, primary menu fields
ASIS 208, 210
auto display pgm 208
CAPS 208, 210
command 208
listing file ID 208
member 208
process options 208
process statements ID 208
search file ID 208
search string fields 208
selection list 208
SuperC side-by-side listing 232
SuperC update files
UPDCMS8 295
UPDCNTL 296, 297, 298
UPDLDEL 299, 300
UPDMVS8 300
UPDPDEL 301
UPDREV 293, 294
UPDREV2 Revision File (2) 294
UPDSEQ0 301, 302
UPDSUMO 302, 303, 304, 305
SWUSIZE 127
Symbol Where Used (SWU) report 150
sample report 150
syntax notation, description v—vii
SYSIN DD statement for ASMXREF 112

T

tabs
See ASMPUT tabs
Tagged Source Program (TSP) report 155
target node 93
three-dimensional nodes 93
token statement file in CMS 116, 117
token statement file in MVS 109
token statement file in VSE 120
TOKEN statement in ASMXREF 128
token statements 128
Token Where Used (TWU) report 154
tokens 129
default 132
file 108, 116, 117
statement 128
topic help 96
transfer file to PC 149
two-dimensional nodes 93

U

ULABL Disassembler statement 51
unit descriptor 140, 141
unit descriptor format 141
unit descriptor keywords
$MAC 141
$MOD 141
$SEG 141
COMP 141
PROD 141
unit name format 141
Unmark All option 84
Unmark option 84
unmarking nodes 84
unresolved external call 93
unresolved nodes 75
UNTIL structured programming macro keyword 29
update file ID field (SuperC primary comparison menu)
See SuperC comparison, primary menu fields
UPDDD SuperC process statement 245, 246
user abends 169
users v
using
ASMXREF 106
ASMXREP 106
using ASMPUT online help 96
using CMS EXECs
ASMXREF 115
invoking ASMXREF in CMS 115
USING Disassembler statement 51
using structured programming macros 11
using the Disassembler 41

V

viewing in ASMPUT
  file information 72
  HLASM Files information 73
  information notebook 72
  Job Id information 73
  Library information 73
  Options information 73
  source code 68
  Statistics information 73
VM publications 338
VSE
  ASMXREF invoking with JCL 120
  ASMXREF sample JCL 121
  ASMXREP EXEC statement 124
  ASMXREP JCL requirements 124
  Disassembler JCL requirements 46
  EXEC ASMXREF statement 123
  SuperC invoking with JCL 199, 221
  SuperC sample JCL 199, 222
VSE publications 338

W

What's This help 97
WHEN macro 12, 37
WHILE structured programming macro keyword 29
window areas 91
  changing size 91
windows
  See ASMPUT windows
working with ADATA files 67
working with the control flow graph 74
WORKSIZE SuperC process statement 262

X

XOBJECT assembly option 64
XRFLANG DLBL statement 123
XRFLANG file in CMS 118
XRFTOKN DLBL statement 123
XRFTOKN in CMS 116

Y

Y2PAST SuperC process statement 266
yellow node 84, 93

Z

z/OS publications
Zoom In icon 86
Zoom In On option 86
Zoom In option 86
Zoom In Rectangle icon 86
Zoom In Rectangle option 86
Zoom Out From option 86
Zoom Out icon 86
Zoom Out option 86
Zoom Out Rectangle icon 86
Zoom Out Rectangle option 86
zoom slider 93
  hiding 86
  showing 86
zooming
  in 86
  out 86

We'd Like to Hear from You

High Level Assembler for MVS & VM & VSE
Toolkit Feature User's Guide
Release 5

Publication No. GC26-8710-08

Please use one of the following ways to send us your comments about this book:

- Mail—Use the Readers' Comments form on the next page. If you are sending the form from a country other than the United States, give it to your local IBM branch office or IBM representative for mailing.
- Electronic mail—Use this Internet ID:
  - Internet: comments@us.ibm.com

  Be sure to include the following with your comments:
  - Title and publication number of this book
  - Your name, address, and telephone number if you would like a reply

Your comments should pertain only to the information in this book and the way the information is presented. To request additional publications, or to comment on other IBM information or the function of IBM products, please give your comments to your IBM representative or to your IBM authorized remarketer.

IBM may use or distribute your comments without obligation.

Readers' Comments

High Level Assembler for MVS & VM & VSE
Toolkit Feature User's Guide
Release 5

Publication No. GC26-8710-08

How satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Technically accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐
Grammatically correct and consistent	☐	☐	☐	☐	☐
Graphically well designed	☐	☐	☐	☐	☐
Overall satisfaction	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

May we contact you to discuss your comments? ☐ Yes ☐ No

Name

Address

Company or Organization

Phone No.

Readers' Comments
GC26-8710-08

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
J87/D325
555 Bailey Avenue
SAN JOSE, CA 95141-9989

Fold and Tape | **Please do not staple** | Fold and Tape

GC26-8710-08

Cut or Fold
Along Line

Program Number: 5696-234

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

High Level Assembler Publications

SC26-4941 *HLASM Programmer's Guide*.
GC26-4943 *HLASM General Information*.
GC26-4944 *HLASM Licensed Program Specifications*.
SC26-4940 *HLASM Language Reference*.
SC26-3494 *HLASM Installation and Customization Guide*.

High Level Assembler Toolkit Feature Publications

GC26-8709 *HLASM Toolkit Feature Interactive Debug Facility User's Guide*.
GC26-8710 *HLASM Toolkit Feature User's Guide*.
GC26-8711 *HLASM Toolkit Feature Installation and Customization Guide*.
GC26-8712 *HLASM Toolkit Feature Debug Reference Summary*.

GC26-8710-08

Spine information:

HLASM Toolkit Feature User's Guide *Release 5*